FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Aplicação Industrial de Técnicas de Análise Causal para Prevenção de Defeitos de Software

Fátima Airosa

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Doutor João Carlos Pascoal Faria

15 de Fevereiro de 2016

Resumo

Empresas de software que têm como objetivo oferecer produtos e serviços com elevada qualidade, com prazos e custos reduzidos, devem aplicar técnicas que permitam minimizar o custo total da qualidade. Quando não se investe na avaliação precoce e na prevenção de defeitos, o custo total da qualidade torna-se muito elevado. A análise causal de defeitos tem se afigurado como sendo uma boa norma na melhoria de processos com base no produto. Sendo este o caminho, e como "mais vale prevenir do que remediar", empresas de maior maturidade têm vindo a apostar na prevenção de defeitos, controlando desta forma o custo/qualidade total do produto. Assim, é apresentado neste documento uma abordagem "*what-to-do*", com o modelo CMMI (*Capability Maturity Model Integration*), que consiste num conjunto de boas práticas a serem seguidas numa perspetiva de melhoria de processo. Um dos princípios mais eficazes e eficientes de prevenção de defeitos baseia-se em "aprender com os erros do passado", ou seja, analisar os erros cometidos com maior impacto negativo, no sentido de identificar as respetivas causas e implementar ações para prevenir (ou detetar mais cedo) a sua ocorrência.

Uma primeira contribuição deste trabalho resultou na identificação e analise das técnicas já existentes na área da análise causal de defeitos. Posteriormente, foram selecionadas as mais adequadas para aplicar este método nas duas empresas selecionadas com vista na prevenção de defeitos durante o ciclo de desenvolvimento de um produto de software. Efetuando, deste modo, uma experiência da aplicação desses procedimentos para avaliar os processos de desenvolvimento usados pelas duas empresas. A pareceria para este projeto contou com a cooperação da *Critical Manufacturing* e do Projeto SIGARRA, que disponibilizaram registos de defeitos para análise.

Uma segunda contribuição foi a apresentação de ações de melhoria e de sugestões a serem implementadas nas duas parcerias, de modo a reduzir o impacto dos defeitos encontrados ao longo de todo o processo de desenvolvimento dos respetivos produtos.

Abstract

Software companies that strive to offer products and services with high quality, on tight deadlines and reduced costs, shall apply techniques to minimize the total quality cost. When there is no investment on early evaluations and defect preventions, the total quality cost becomes unbearable. The causal analysis of defects has established itself as a good standard in process improvement based on the product. This being the right way, and since "it's better safe than sorry", more mature companies have been focusing on defect prevention, controlling the product's total cost and quality. Therefore, in this paper it is presented an approach "*what-to-do*", with the model CMMI (*Capability Maturity Model Integration*), which consists of a set of best practices to follow when the process improvement is at stake. One of the most effective and efficient principles to prevent defects is based on "learning from previous mistakes , that is, to analyse the mistakes made with the highest negative impact, in order to identify their causes and implement measures to prevent (or to detect earlier) its future occurrence.

The first contribution of this project is a revision and analysis of the currently existing techniques on causal analysis of defects. Subsequently, the most suitable were selected to apply this method in the two selected companies in order to prevent defects during the development cycle of a software product, performing this way an experience on the application of these procedures in order to evaluate the development processes used by the two companies. This project's partnership counted on the cooperation of *Critical Manufacturing* and SIGARRA Project, which provided defect records for analysis.

A second contribution of this project was the presentation of improvement measures and suggestions to be implemented by the two partnership companies in order to reduce the impact of defects found along the development process of their respective products.

Para a minha avó:

"*Somehow I know we'll meet again. Not sure quite where and I don't know just when. You're in my heart, so until then it's time for saying goodbye.*"

Muppets

"Why spend all this time finding, fixing, and fighting
when you could have prevented the problem in the first place?"

Philip B. Crosby

Conteúdo

Lista de Figuras

Abreviaturas e Símbolos

AC	Análise Causal
ACD	Análise Causal de Defeitos
AP	Área de Processo
API	Application Programming Interface
CAR	*Causal Analysis and Resolution*
CE	Causa-efeito
CL	*Capability Level*
CM	*Critical Manufacturing*
CMMI	*Capability Maturity Model Integration*
CoQ	*Cost of Quality*
CRSCUP	Centro de Recursos e Serviços Comuns da Universidade do Porto
DCA	*Defect Causal Analysis*
DMADV	*Define, Measure, Analyze, Design, Verify*
DMAIC	*Define, Measure, Analyze, Improve, Control*
GA	Gestão Académica
GC	*Generic Goal*
GI	Gestores de Informática
GRH	Gestão de Recursos Humanos
GUI	*Graphical User Interface*
ISO	*Internationl Organization for Standardization*
MES	*Manufacturing Execution System*
ML	*Maturity Level*
MQM	*Modeling for Quality Management*
ODC	*Orthogonal Defect Classification*
SI	Sistema de Informática
SIGARRA	Sistema de Informação para a Gestão Agregada dos Recursos e dos Registos Académicos
SP	*Specific Pratice*
SPC	*Statistical Process Control*
TDD	*Test Driven Development*
TFS	*Team Foundation Server*
TQM	*Total quality management*
UI	*User Interface*
UO	Unidades Orgânicas
UP	Universidade do Porto
WIP	*Work in Progress*

Capítulo 1

Introdução

1.1 Enquadramento

No âmbito da Dissertação do curso MIEEC da FEUP, em parceria com a CM e com o SIGARRA, foi realizado o estudo com o título Aplicação Industrial de Técnicas de Análise Causal para Prevenção de Defeitos de Software. Este trabalho pretende colocar em prática essas técnicas nas empresas colaboradoras.

A Critical Manufacturing (CM) é uma empresa que proporciona às indústrias de produção discreta uma solução de gestão e controlo de produção que habilita as unidades fabris para alcançarem os seus objetivos. A CM é responsável por detalhar as operações, facilita a visibilidade sobre os processos e custos refletidos na cadeia de abastecimento e adapta a implementação nas infraestruturas já existentes. Como resultado, os produtos que a CM desenvolve facilitam a redução contínua de custos, a flexibilidade necessária para satisfazer a procura e, sobretudo, capacitam os clientes com uma maior agilidade, visibilidade e fiabilidade [1]. A Critical Manufacturing possui certificados na área da qualidade, tais como, ISO9001 e nível 3 do CMMI.

O projeto SIGARRA encontra-se responsável pela gestão e manutenção do sistema de informação da Universidade do Porto, sendo, por isso uma entidade sem fins lucrativos. Esta plataforma constitui-se como um serviço que disponibiliza informação relevante sobre as atividades da instituição académica, e uma infra-estrutra que funciona como uma espinha dorsal em que podem ser interligados outros subsistemas [2].

São duas empresas que se encontram em posições opostas no que toca a organização e a gestão da qualidade, mas ambas com o mesmo objetivo, prevenir e reduzir os defeitos nos seus produtos e, deste modo, aumentar a qualidade e a confiança nas soluções de software que desenvolvem.

1.2 Motivação

"The mason who builds a house which falls down and kills the inmate shall be put to death."
(Código de *Hammurabi*, governante de Babilónia)

A preocupação com a qualidade não é algo novo. O controlo da qualidade total nasceu já nos primórdios da civilização humana, com os historiadores a retroceder até 3000 A.C., como se pode verificar através da citação do governante da Babilónia [3]. As organizações preocupam-se cada vez mais em oferecer aos seus clientes produtos e serviços com qualidade.

A qualidade é um aspeto bastante importante e crítico no desenvolvimento de software, no entanto, é muitas das vezes ignorada durante a fase de produção. A maioria dos defeitos são apenas identificados na fase de testes, na qual os custos para a sua correção são elevados ou as metodologias aplicadas não são suficientes [4]. Na ausência da aplicação destas práticas, o que resulta é um desperdício de recursos humanos e materiais por parte das empresas, assim como uma perda de confiança por parte do cliente, causando, deste modo, um impacto negativo nas receitas da empresa. Aplicação de processos que promovem a deteção contínua e precoce de defeitos torna o projeto menos dispendioso.

Por fim, a qualidade ajuda a fidelizar o cliente, que é a grande fonte de proveito de uma organização. Joseph Juran evidência a existência de um sujeito, que vai receber um produto ou serviço, cujas as necessidades de uso precisam de ser satisfeitas [5]. Se o produto ou serviço não se comportar como o esperado, cumprindo os requisitos estabelecidos pelo cliente, este perderá a confiança no produto e na organização que o desenvolveu [5]. Desta forma, uma empresa que aposte na qualidade consegue fidelizar um maior número de clientes, conseguindo uma maior estabilidade nos mercados económicos.

1.3 Objetivos

Este projeto foi o realizado com vista na melhoria de processos e na caracterização da utilização do método de análise causal de defeitos. Para tal, foram analisadas as abordagens, os modelos e as práticas de melhoria de processos através do produto desenvolvido.

O grande objetivo deste projeto é demonstrar a aplicabilidade das técnicas de análise causal de defeitos em duas organizações. Foram, então, escolhidas duas organizações que cederam os seus registos de defeitos para análise. Análise essa que conta com seis passos, desde a seleção de dados, classificação, identificação de erros e das suas causas, até à elaboração de um plano que contém propostas de ações de melhoria a serem implementadas nas empresas selecionadas.

Os objetivos desta dissertação passam, também, por identificar os benefícios e dificuldades subjacentes à sua aplicação

1.4 Estrutura do documento

O presente relatório está divido em cinco capítulos. O primeiro capítulo denomina-se Introdução e apresenta uma informação geral sobre a dissertação, onde estão contemplados o contexto deste trabalho, o problema proposto e a motivação para a sua resolução, os objetivos e resultados esperados e, por fim, a forma de como está estruturado todo o documento.

O segundo capítulo chama-se Revisão Bibliográfica e aborda temas importantes tais como, a gestão da qualidade e o seu custo durante o processo de execução do produto. É feita também uma pequena apresentação sobre análise causal e *Capability Maturity Model Integration* (CMMI), técnicas para identificar e analisar defeitos específicos, de modo a prevenir que estes ocorram no futuro e, por último, uma pequena conclusão com uma síntese de todo o capítulo.

O terceiro capítulo intitula-se Aplicação na Critical Manufacturing. Neste capítulo é feita uma descrição do sistema de qualidade existente na CM e é feita a análise causal de defeitos ao produto cmNavigo.

O quarto capítulo denomina-se Aplicação no Projeto SIGARRA. É feita uma breve descrição do sistema de qualidade no projeto SIGARRA e é conduzida a análise causal de defeitos à plataforma SIGARRA.

O quinto capítulo, Conclusão, contém as conclusões acerca do trabalho efetuado e os resultados obtidos.

Capítulo 2

Revisão Bibliográfica

Neste capítulo são apresentados os conceitos necessários para uma melhor compreensão do tema apresentado e a forma como estes se relacionam. Consiste na descrição dos vários temas que foram necessários explorar e estudar para o desenvolvimento desta dissertação.

2.1 Gestão da qualidade

O conceito de qualidade total como uma estratégia de negócio começou a crescer e amadurecer nos Estados Unidos no final de 1980 e início de 1990 [6]. Contudo, vários elementos da qualidade total, tais como, o controlo estatístico de processos, ferramentas de resolução de problemas, serviço ao cliente e a documentação de processos, foram usados durante muitos anos pelas empresas, mas separadamente. A gestão da qualidade total (*Total quality management* - TQM) é a fusão de todas as funções e processos dentro de uma organização, com o propósito de atingir uma melhoria contínua dos seus produtos e serviços. A satisfação do cliente é o principal objetivo [6].

A TQM "obriga"as empresas a pensar em todas as suas funções, e a planear desde o início do processo até à sua conclusão. Deste modo, e independentemente da conjuntura da empresa, consegue-se interligar todas as funções dos diferentes níveis da organização. Deste modo, a eficácia do sistema global é sempre superior à eficácia individual dos seus subsistemas [6]. Os subsistemas incluem todas as funções organizacionais do ciclo de vida de um produto, tais como: projeto, planeamento, produção, distribuição e serviço de campo. Os subsistemas de gestão também exigem a integração da estratégia com foco no cliente, as ferramentas de qualidade, e do envolvimento dos trabalhadores [6].

Em Maio de 1990, numa conferência internacional, foram resumidas questões chaves e terminologias relacionadas com a TQM [6]:

- Custo da qualidade
- Mudança cultural
- Ativação de mecanismo de mudança

- Implementação TQM
- Gestão de comportamento

A conclusão mais relevante alcançada nesta conferência foi que qualquer produto, processo ou serviço pode ser melhorado, e uma organização de sucesso é aquela que procura e insiste em novas oportunidades de melhoria contínua em todos os níveis.

2.1.1 Custo da qualidade

"Quality is measured by the cost of quality which is the expense of non conformance — the cost of doing things wrong."(*Philip Crosby, Quality is Free*)

Reduzir o custo da qualidade, *Cost of Quality* (CoQ), é agarrar a oportunidade de aumentar os lucros sem aumentar as vendas, sem investir em novos equipamentos, e sem contratar novas pessoas. Para medir este custo é necessário analisar e explorar o desperdício. O primeiro passo é recolher todos os custos envolvidos na operação da empresa, nomeadamente, todos os desperdícios, todos os materiais devolvidos ainda em fase de produção e na de pós-venda - garantia, lidar com as reclamações, realizar inspeções e testes, e por fim todos os custos de erros, como por exemplo, alterações nas ordens de compra ou de processos de engenharia [7]. Para que haja uma melhor perceção dos custos, o CoQ pode ser medido usando a percentagem de vendas ou a percentagem dos custos de vendas [7]. No caso da *Hewlett-Packard*, por não fazer as coisas certas logo à primeira, estimou um impacto negativo nas suas receitas entre 25 a 30% [8]. Após a recolha de todos os dados necessários e de calculados os CoQ, ou uma aproximação, podem ser delineados os objetivos para reduzir estes custos [7]. Encontrar as respostas para a má qualidade não é de todo uma tarefa fácil, no entanto, o esforço que as empresas fazem para encontrar a resposta acaba por compensar. A Motorola conseguiu reduzir o custo da má qualidade num total de 5% das suas vendas, o que equivale a aproximadamente 444 milhões de euros por ano [8].

Os responsáveis pelas análises do CoQ frequentemente perguntam-se "Quais os custos que devem ser incluídos?"Para responder a esta questão, Crosby criou uma lista de custos que auxilia a execução desta tarefa, no entanto esta lista deve ser ajustada consoante o negócio da empresa. Podem-se agrupar os custos em três categorias [7]:

- Custos de prevenção (*prevention costs)*: custos de todas as atividades envolvidas em prevenir defeitos de conceção e de desenvolvimento, compra e outros aspetos no processo de início ou de criação de um novo produto ou serviço. Envolvem também ações preventivas conduzidas durante o ciclo de negócio.
- Custos de avaliação (*appraisal costs)*: custos de todas as atividades que verificam, durante as inspeções, testes, e outros planos de avaliação ao hardware, software ou serviços que estes se encontram de acordo com os requisitos. Estes incluem especificações de marketing ou clientes, assim como documentação e informação sobre processos de engenharia.

- Custos de falhas (*failure costs)*: custos associados a certas especificações que não foram cumpridas conforme os requisitos. Estas falhas estão relacionadas com aspetos de avaliação, disposição e consumo. Estão também incluídos os custos de todo o material e mão-de-obra envolvida na conceção. Esta categoria de custos pode ser dividida em duas [8], custo de falhas internas (*internal failure costs)* e custo de falhas externas (*external failure costs)*. A primeira diz respeito a falhas ocorridas durante o processo de produção; já as falhas externas estão relacionadas a defeitos encontrados pelo cliente.

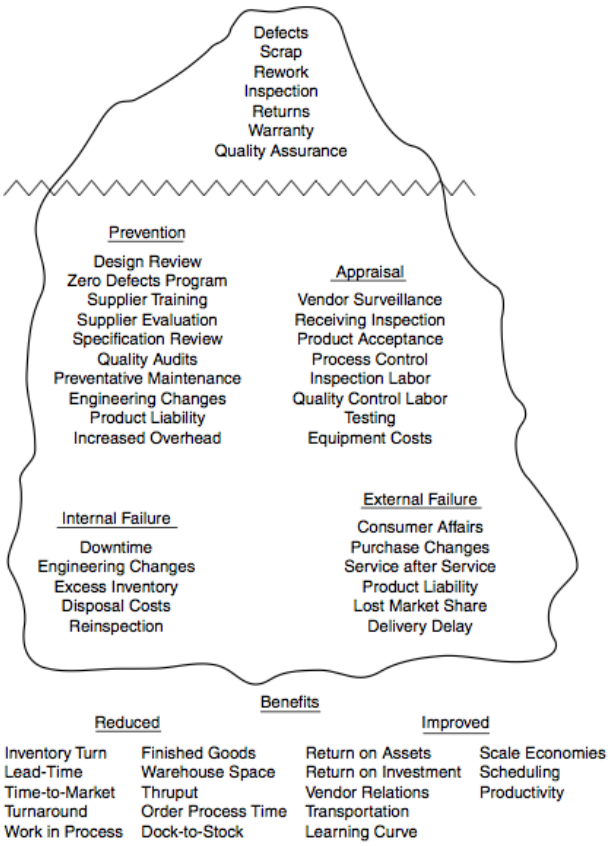

Figura 2.1: Controlo do custo da qualidade

Na figura 2.1 [8] os vários custos da qualidade estão agrupados pelas quatro categorias já referidas anteriormente. A imagem é uma analogia a um *iceberg*, visto que apenas 10% dos problemas de qualidade se encontram visíveis [8]. De entre todas as categorias apresentadas, a prioridade deve ser dada à que envolve os custos de prevenção, uma vez que, regra geral, fica mais barato prevenir um defeito do que corrigí-lo mais tarde. O motivo pelo qual essa categoria deve ter uma especial atenção está ilustrado na figura 2.2 [8]: ao investir na prevenção economiza-se na correção de erros e no tratamento de falhas.

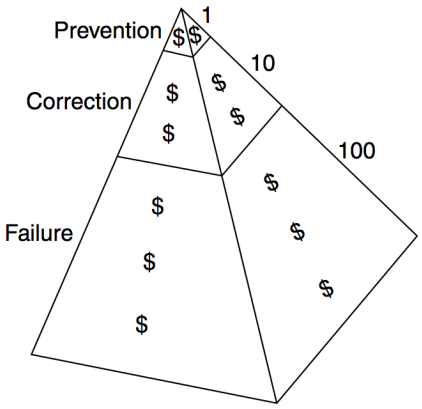

Figura 2.2: Regra 1-10-100

Um bom exemplo da regra 1-10-100, demonstrada na figura 2.2, foi dado por Richard W. Anderson, diretor geral da *Hewlett-Packard's* [9]:

"If you catch a two cent resistor before you use it and throw it away, you lose two cents. If you don't find it until it has been soldered into a computer component, it may cost $10 to repair the part. If you don't catch the component until it is in the computer user's hands, the repair will cost hundreds of dollars. Indeed, if a $5000 computer has to be repaired in the field, the expense may exceed the manufacturing cost."

À medida que o erro vai avançando nos vários processos da linha de produção sem ser detetado, pode evoluir para defeito e, mais tarde, provocar uma falha.

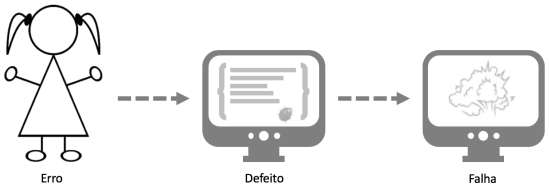

Figura 2.3: Erro - Defeito - Falha

Um erro resulta de uma ação humana que produz um resultado incorreto, que por sua vez origina um defeito no software; este defeito progride para uma falha, um desvio no comportamento do software para o qual foi criado [10]. Em suma, quanto mais próximo do produto final for detetado o problema, maior o custo da falha provocada e maior o custo de correção [9]. Assim sendo, torna-se necessário uma maior aposta na prevenção de defeitos, conseguindo deste modo um melhor controlo na qualidade dos produtos e evitar os gastos provenientes desses mesmo defeitos no sistema de produção.

2.1.2 Qualidade do produto *versus* qualidade do processo

Se um produto cumprir as expectativas de um cliente, este ficará satisfeito e considerará que o produto tem qualidade. No entanto, se as suas expectativas não forem cumpridas, o cliente irá classificar o produto como sem qualidade. Isto significa que a qualidade do produto pode ser definida como "a capacidade de satisfazer as necessidades e expectativas dos clientes"[11]. A qualidade precisa, então, de ser definida em primeiro lugar como o cumprimento de requisitos ou características que variam consoante a necessidade do cliente e o produto.

Uma das abordagem para controlar a qualidade do produto baseava-se apenas na inspeção do mesmo no seu estado final. Esta era uma alternativa bastante rigorosa que permitia selecionar o bom do mau produto. No entanto, esta metodologia é bastante cara e gera imenso desperdício, já que é feita a separação do produto depois de todo o seu processo de produção estar concluído, consumindo recursos e bens materiais desnecessariamente [5]. Uma forma de reduzir os desperdícios e defeitos encontrados no produto final, e consequentemente, melhorar a sua qualidade, consiste na utilização de processos adequados de verificação e validação ao longo do ciclo de produção [12]. Se um processo estiver bem planeado e se for bem implementado, o produto será fabricado de acordo com as especificações [5]. Para melhorar a qualidade do processo foram criadas definições precisas e mensuráveis para o controlo da qualidade, assim como foram desenvolvidas técnicas estatísticas para avaliar a produção e, consequentemente, melhorar a qualidade.

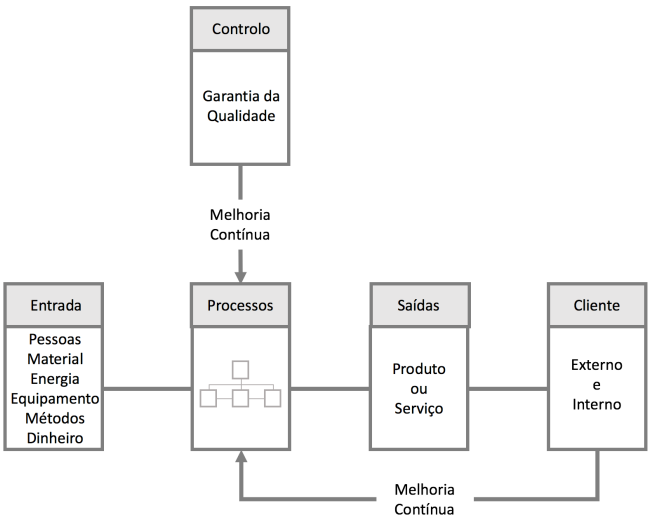

Figura 2.4: Sistema de Gestão de Qualidade[3]

Como já foi referido anteriormente, se se focar as atenções apenas no controlo das saídas do sistema, ou seja, depois da transformação estar completa, nada pode ser feito para corrigir os defeitos. O mesmo pode ser comprovado quando o produto ou serviço é inspecionado e acaba por ser rejeitado por não cumprir os requisitos. O que é necessário, tal como demonstrado na figura 2.4, é um sistema de prevenção doe defeitos que visa a melhoria dos processos [3]. Desta forma,

aposta-se na prevenção dos defeitos ao identificar-se e remover-se as suas causas, resultando na melhoria contínua da operação da empresa.

A gestão da qualidade utiliza, portanto, o controlo dos processos e do WIP (*Work in Progress*) para garantir a qualidade do produto final ou serviço. A norma ISO 9000, o modelo CMMI e as técnicas do *Six Sigma* são alguns dos métodos de gestão e técnicas que impulsionaram a melhoria da qualidade ao longo dos anos, e que ainda são usadas nos dias de hoje.

2.1.3 Referenciais ou normas

ISO 9001:2015

ISO 9001:2015 (*International Organization for Standardization*) faz parte de um conjunto de três normas internacionais para sistemas de gestão da qualidade (*Quality Management Systems* - QMS). Estas normas especificam os requisitos e recomendações para o planeamento e avaliação de sistemas de gestão [13], auxiliam as organizações a implementar e operar sistemas eficazes na gestão da qualidade. Estes padrões fornecem princípios de gestão da qualidade aprovados por um comité internacional [13]. As três normas da família ISO 9000 são:

- ISO 9000: QMS - Princípios e vocabulário
- ISO 9001: QMS - Requisitos
- ISO 9004: QMS - Procedimentos para melhoria de desempenho

A norma ISO 9000 tem como objetivo proporcionar uma introdução aos princípios da gestão da qualidade e uma explicação da terminologia utilizada na família ISO. A ISO 9001 tem como propósito ajudar as organizações a fornecer produtos e serviços que atendam às necessidades e expectativas dos clientes, cumprindo com todos os requisitos. A ISO 9004 fornece orientações para melhorar a eficiência, eficácia e desempenho global de uma organização [13].

A norma ISO 9001 baseia-se em sete princípios de gestão da qualidade. Estes princípios devem ser utilizados como uma estrutura para orientar as organizações e empresas para um desempenho melhor [14]. Os sete princípios de gestão da qualidade são [15]:

1.*Customer focus*

2. *Leadership*

3.*Engagement of people*

4.*Process approach*

5.*Improvement*

6.*Relationship management*

7.*Evidence-based decision making*

Na figura 2.5 está representada a estrutura da norma ISO 9001:2015 (recente versão da norma). É composta por um número de secções diferentes, cada uma focada nos requisitos envolvidos nos diferentes aspetos de um sistema de gestão da qualidade [15].

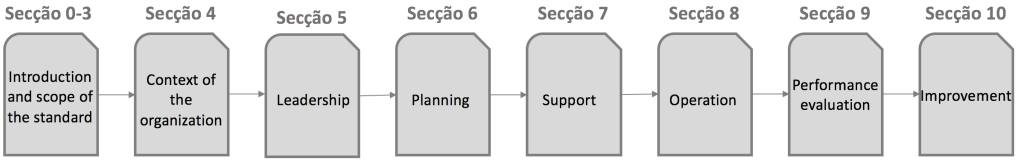

Figura 2.5: Estrutura ISO 9001:2015

SIX SIGMA

A abordagem *Six Sigma* consiste na aplicação sistemática de técnicas e conceitos estatísticos com vista a reduzir as variações no processo e, deste modo, prevenir defeitos no produto. Não é apenas uma técnica, é também uma filosofia baseada no desejo de eliminar as perdas e melhorar o desempenho. O nome *Six Sigma* é derivado do conceito estatístico do desvio padrão; a variação de um processo é tipicamente medida em números de desvios padrões da média [16]. Neste caso o processo produz apenas 3,4 defeitos por milhão de oportunidades (*Defects Per Million Opportunities* - DPMO), portanto, o Six Sigma pode ser visto, também, como uma meta, onde os processos não só encontram menos defeitos, como também o fazem de forma mais consistente (baixa variabilidade) [16]. Esta abordagem reduz a variação, fazendo com que os produtos ou serviços possam ser entregues conforme o esperado e de uma maneira mais confiável.

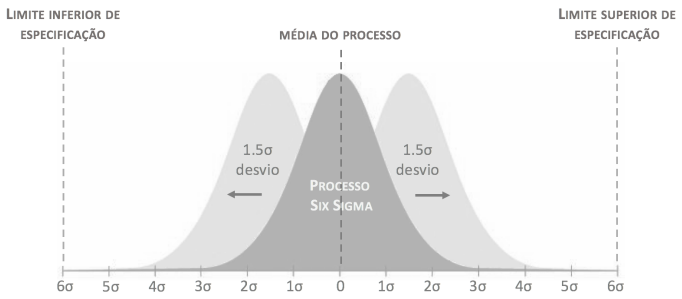

Figura 2.6: Gráfico *Six Sgima*

Analisando agora a figura 2.6, o conceito *Six Sigma* consiste em minimizar o número de itens defeituosos, de tal forma que seis desvios-padrão (seis sigma) seja a maior variação entre a média do processo e os seus limites de especificação inferior e superior. *Six Sigma* também considera a degradação do processo ao longo do tempo, e por esse motivo pode tolerar uma mudança nos desvios padrão e ainda manter uma margem de segurança entre a média do processo e os seus limites de especificação [16].

O *Six Sigma* é composto por duas metodologias, cada uma contendo cinco fases.

- DMAIC (*define, measure, analyze, improve, control*) deve ser usada quando um produto ou processo existe na empresa, mas não está a cumprir as especificações do cliente ou não está a funcionar correctamente [17].

Define: Definir os objetivos do projeto e dos clientes

Measure: Medir o processo para determinar o desempenho atual

Analyze: Analisar e determinar a causa raiz dos defeitos

Improve: Melhorar o processo de eliminação de defeitos

Control: Controlar futuro desempenho do processo

- DMADV (*define, measure, analyze, design, verify*) é usada quando um processo ou produto não existe e precisa de ser criado, ou o produto ou processo já existe e foi melhorado, mas no entanto não cumpre com a performance requerida pelo *Six Sigma* [17].

  Define: Definir os objetivos do projeto e dos clientes

  Measure: Medir e determinar as necessidades e especificações do cliente

  Analyze: Analisar as opções de processo para atender às necessidades dos clientes

  Design: Estrutura o processo para atender às necessidades dos clientes

  Verify: Verifica o desempenho do projeto e a capacidade de corresponder com as necessidades dos clientes

CAPABILITY MATURITY MODEL

O SEI (*Software Engineering Institute*) a pedido do departamento de defesa dos Estados Unidos da América, elaborou uma ferramenta que permite às empresas evoluir os seus processos. O departamento de defesa, sendo um grande cliente de sistemas de software, necessitava de um sistema que lhe permitisse avaliar a qualidade do produto que adquiria às empresas de software, e se esse mesmo produto correspondia às suas necessidades. O *Capability Maturity Model* (CMM) foi, então, concebido como uma ferramenta de avaliação dos processos de uma empresa para avaliar a sua capacidade em fornecer ou não o produto desejado [18].

O nível de maturidade de um processo de desenvolvimento reflete em que medida a empresa é capaz de fornecer produtos de baixo custo e com alta qualidade. Assim sendo, o CMM é um quadro de avaliação de maturidade dos processos atuais de uma empresa. Com base nos resultados da avaliação, a empresa pode empenhar-se em melhorar os seus processos para alcançar o próximo nível de maturidade do CMM [19].

Uma organização imatura pode não ter processos definidos e, mesmo que os tenha, a empresa pode não segui-los. Nestes casos, normalmente os funcionários reagem aos problemas quando estes ocorrem, em vez de tomarem medidas preventivas para eliminar ou reduzir a frequência das suas ocorrências. Além disso, os problemas dos produtos e dos processos são resolvidos à medida que vão aparecendo, deste modo as estimativas de custo, os cronogramas e a qualidade são altamente imprecisos devido à completa inexistência de um programa de avaliação da qualidade do produto ou processo. Por outro lado, uma organização madura tem as suas atividades devidamente

planeadas. Tanto os processos como os produtos são avaliados para acompanhar o progresso e a qualidade e as estimativas são mais precisas devido ao cumprimento de um programa de avaliação. Adicionalmente, os funcionários são mantidos a par dos novos desenvolvimentos e através de formações é feito um esforço contínuo para melhorar a qualidade dos processos e produtos, reduzir os custos e os prazos de entrega. Estes processos definidos são continuamente atualizados para tirar proveito de experiência de projetos anteriores. É desta forma que uma organização amadurece: gradualmente; ou seja, os processos são melhorados através de uma abordagem evolutiva, em vez de serem efetuadas mudanças drásticas [20].

A figura 2.7 representa o conceito de nível de maturidade e cada nível é constituído por áreas-chaves de processo específicas (*key process areas*). Cada área-chave conduz a metas de melhorias do processo e encontra-se organizada em características comuns (*common features*), que determinam as práticas-chave a implementar (*key practices*).

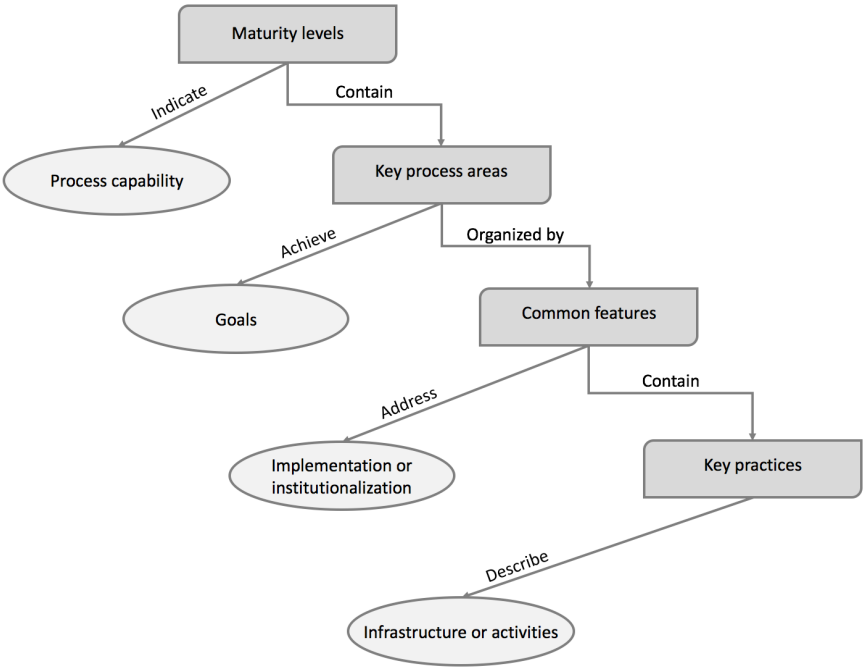

Figura 2.7: Estrutura do CMM

Sucedendo os resultados impressionantes do CMM na avaliação de empresas, surgiu o CMM para o desenvolvimento de software, conhecido como CMM-SW (software - SW). Depois de algumas versões, muitas empresas de software começaram a utilizar esta ferramenta para realizarem autoavaliações e avaliações externas.

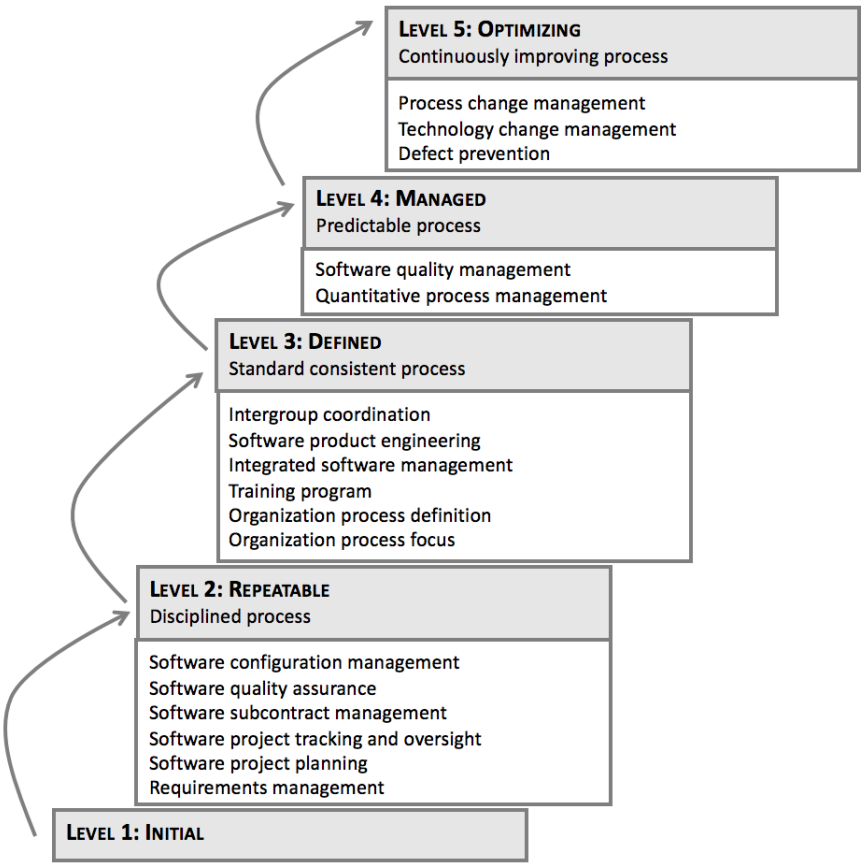

Figura 2.8: Níveis de maturidade do CMM-SW

O CMM-SW é composto por cinco níveis de maturidade (figura 2.8), cada um com as suas próprias características. O nível 1 corresponde a organizações mas imaturas, onde não existe nenhum tipo de metodologia implementada e tudo é feito de forma desorganizada. O último nível pertence a organizações mais maduras, em que cada detalhe do processo está definido, objetivos foram delineados, o processo é quantificado, acompanhado e alterado sem prejudicar o desenvolvimento. Para atingir o nível 5, é necessário obter a maturidade no nível 4, que por sua vez exige a do nível 3, e assim sucessivamente. À medida que a organização passa de um nível para o outra a capacidade do processo melhora, resultando num produto de melhor qualidade a um custo de produção menor [20].

2.2 CMMI

Após o desenvolvimento e a aplicação bem sucedida do CMM na área do software, conhecido como CMM-SW, foram também desenvolvidos outros CMMs para outras áreas. O CMM é um modelo de referência de práticas maduras, numa disciplina específica, utilizado para melhorar a capacidade de um grupo executar essa disciplina. Tal como foi explicado anteriormente, o conceito

CMM não é específico para software, existe uma grande diversidade na aplicabilidade do CMM [20]. Alguns dos exemplos de modelos de CMM para diferentes áreas são:

- *Software CMM*
- *Systems engineering CMM*
- *Integrated product development CMM*
- *People CMM*
- *Supplier source CMM*

Dada a diversidade do CMM, surgiu uma necessidade de ter uma visão unificada de melhoria de processos em toda a empresa, surgindo assim o *Capability Maturity Model Integration* (CMMI). O CMMI engloba informações do modelo *Software CMM*, *Systems engineering CMM* e *Integrated product development CMM* com a intenção de fornecer um CMM que abrange os desenvolvimentos de produtos e de serviços e manutenção, assim como uma estrutura extensível, de modo a serem adicionados novos conhecimentos [21].

2.2.1 Componentes do CMMI

O propósito do CMMI é controlar a seleção e utilização de componentes dos vários modelos CMM para as diversas áreas de interesse. Aquando da construção de um novo modelo CMMI são usados componentes já existentes que atendam às necessidades da nova área de interesse. Usando esses componentes, reduz-se a quantidade de formação necessária e o tempo de adaptação dos processos já praticados. A figura 2.9 mostra os vários componentes (*required, expected*, e *informative*) que podem ser encontrados nos modelos CMMI.

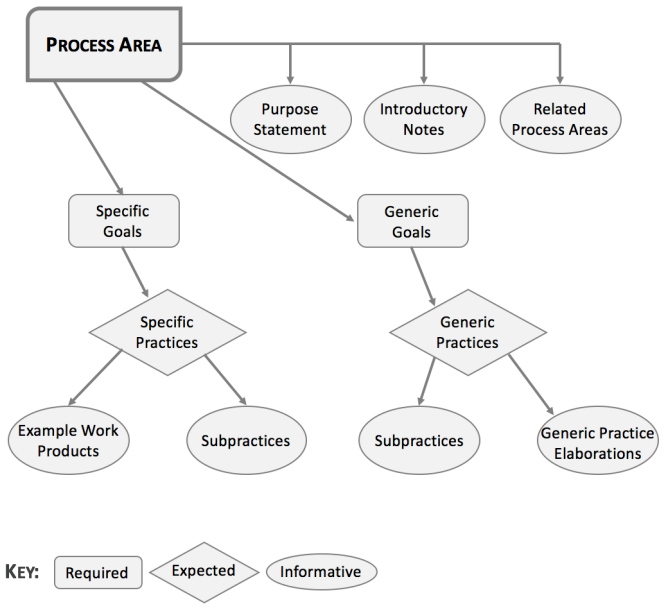

Figura 2.9: Componentes do CMMI

- *Process Area*: é um conjunto de práticas relacionadas com uma área que, quando implementadas em conjunto, satisfazem um conjunto de metas consideradas importantes para melhorar essa mesma área [22].
- *Required*: componentes essenciais para a melhoria no processo de uma determinada área; decidem se o objetivo numa determinada área de processo foi satisfeito [22].
  - *Specific Goals*: descreve características únicas que devem estar presentes para satisfazer a área de processo.
  - *Generic Goals*: são objetivos genéricos aplicados a várias áreas do processo; descrevem características que devem estar presentes para institucionalizar uma área de processo.
- *Expected*: componentes que descrevem as atividades que são importantes na obtenção de um componente necessário (*Required*); guiam na implantação de melhorias ou na execução de avaliações [22].
  - *Specific Practices*: é a descrição de uma atividade que é considerada importante para alcançar o objetivo especifico (*Specific Goals*) associado [22].
  - *Generic Practices*: são práticas genéricas que são aplicadas a várias áreas de processo, e descrevem as práticas importantes associadas a um objetivo genéricos (*Generic Goals*) [22].
- *Informative*: componentes que ajudam a compreender os componentes *Required* e *Expected* do modelo CMMI; podem ser uma explicação detalhada ou informação útil [22].
  - *Subpractices*: é uma descrição detalhada que fornece orientação para interpretar e implementar uma prática especifica ou genérica (*specific practice* e *generic practice*); um componente meramente informativo destinado a fornecer ideias que podem ser úteis na melhoria de processos [22].

De modo a facilitar a construção de modelos que preservassem o legado do CMM e dos modelos CMMI, os componentes foram agrupados em três constelações [23]:

- *CMMI for Acquisition* (CMMI-ACQ): fornece um conjunto integrado e abrangente de diretrizes para a aquisição de produtos e serviços; o foco deste modelo é nos processos de aquisição [24].
- *CMMI for Development* (CMMI-DEV): proporciona orientação para a aplicação de melhores práticas para o desenvolvimento de produtos e serviços de qualidade para satisfazer as necessidades dos clientes [22].
- *CMMI for Services* (CMMI-SVC): proporciona orientação para a aplicação de melhores práticas do CMMI numa organização de prestação de serviços; práticas focadas em atividades para fornecer serviços de qualidade aos clientes e utilizadores finais. Integram conhecimentos que são essenciais para um prestador de serviços [25].

A interseção das três constelações envolve 16 áreas de processo (figura 2.10); estas áreas de processo abrangem conceitos básicos que são fundamentais para a melhoria de processos nos três ramos de interesse (aquisição, desenvolvimento ou serviços). Algumas das áreas de processo são usadas em todas as constelações, no entanto as restantes podem ser ajustadas para lidar com uma área especifica [26].

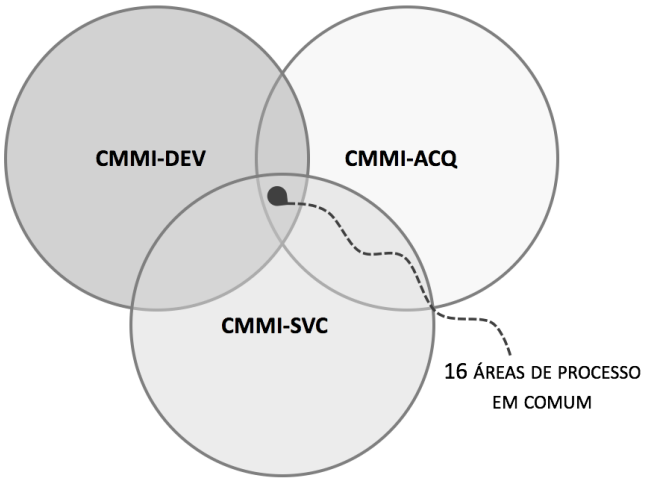

Figura 2.10: Áreas de processo em comum nas três constelações

Capability Maturity Model Integration é um conjunto de boas práticas que as empresas tecnológicas devem seguir, de modo a serem capazes de melhorar a qualidade dos seus processos, e deste modo, forçando-as a uma melhoria contínua dos produtos desenvolvidos. As empresas que adotam este tipo de modelo são também desafiadas a atingir um certo nível de qualidade. O CMMI é compostos por três modelos diferentes (CMMI-AQC, CMMI-DEV e CMMI-SVC). No total, o CMMI engloba 24 áreas de processo (AP) [25].

2.2.2 Estrutura do CMMI-DEV

O CMMI-DEV é divido em duas representações (contínua ou por estágios), e cada uma delas é composta por cinco níveis de maturidade; dentro de cada nível encontram-se as respetivas áreas de processo [25].

REPRESENTAÇÃO CONTÍNUA E POR ESTÁGIOS

O CMMI suporta dois tipos de representações, a representação contínua e a representação por estágios. Na primeira, temos um tipo de modelo mais flexível, onde a empresa pode escolher quais as áreas de processo a serem exploradas e melhoradas. É necessário no entanto, ter em atenção que existem algumas áreas de processo que são dependentes de outras, isto é, para ser implementada uma certa área de processo é necessário cumprir os objetivos impostos pela área de processo precedente [22].

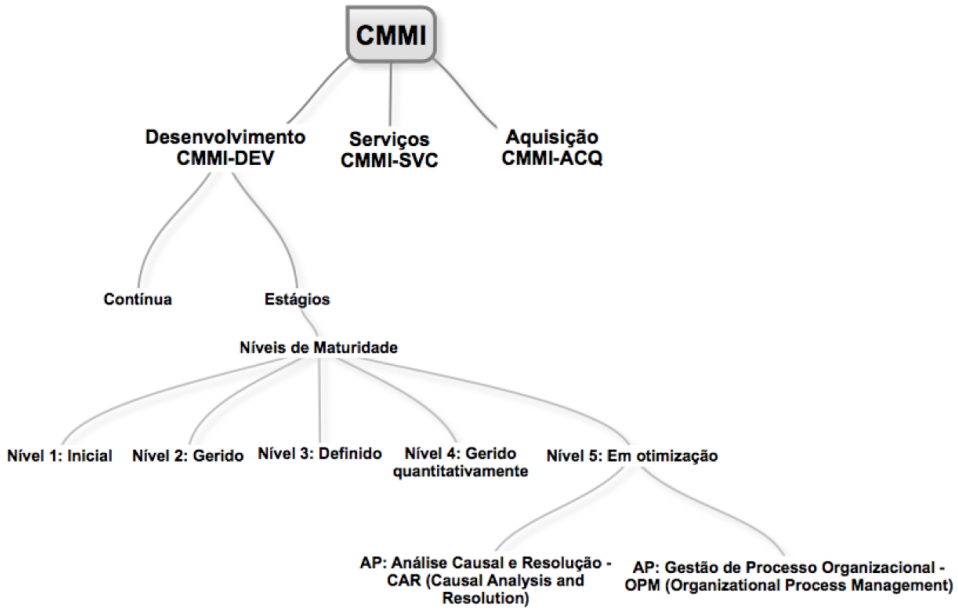

Figura 2.11: Estrutura do CMMI

Existem quatro níveis de capacidade, (*Capability Levels* - CL), numerados de 0 a 3:

- Nível 0: Incompleto (*Incomplete*)
- Nível 1: Executado (*Performed*)
- Nível 2: Gerido (*Managed*)
- Nível 3: Definido (*Defined*)

Como mostra a figura 2.12, na representação contínua podem ser selecionadas as AP a serem melhoradas. Por exemplo, a empresa pode ter um nível de capacidade 1 na área de processo 4 e um nível de capacidade 3 na área de processo 2, diferentes níveis para diferentes AP.

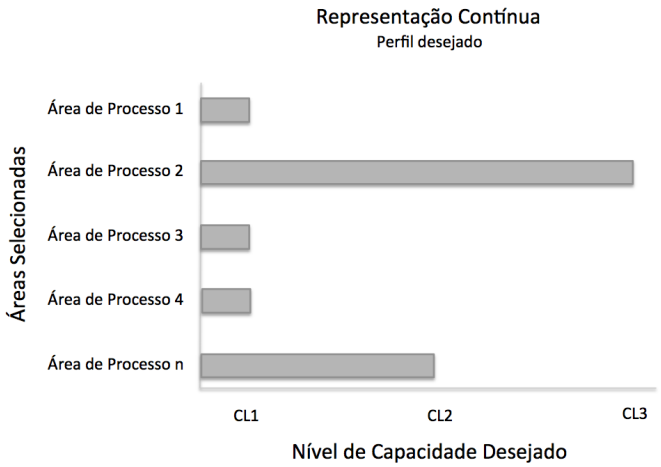

Figura 2.12: Exemplo de áreas de processo na representação contínua

Por outro lado, a representação por estágios obedece a uma sequência de melhoria, cada nível é composto por um grupo de AP que têm de ser aprovadas antes de poder avançar para o nível seguinte, ou seja, o anterior serve sempre de suporte para o próximo. Esta representação é constituída por cinco níveis de maturidade, (*Maturity Levels* - ML), numerados de 1 a 5:

- Nível 1: Inicial (*Initial*)
- Nível 2: Gerido (*Managed*)
- Nível 3: Definido (*Defined*)
- Nível 4: Gerenciado Quantitativamente (*Quantitatively Managed*)
- Nível 5: Em optimização (*Optimizing*)

Através da figura 2.13, na representação por estágios, para ser atingido o nível de maturidade 3, todos os objetivos propostos pelas áreas de processo que estão inseridas no nível de maturidade 2 têm de ser atingidas.

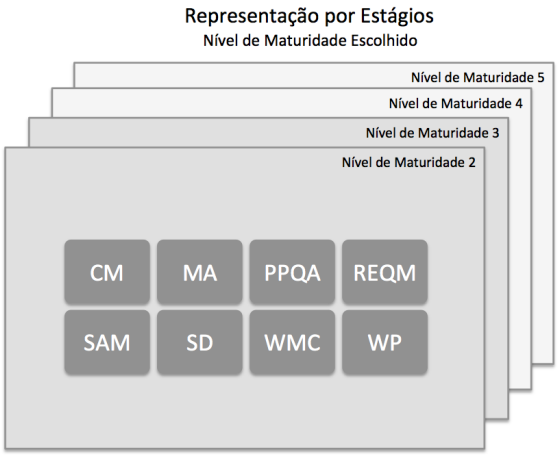

Figura 2.13: Exemplo de áreas de processo na representação por estágio

CINCO NÍVEIS DE MATURIDADE

As áreas de processo estão agrupadas consoante cada nível de maturidade. Para que seja possível alcançar um nível de maturidade superior, é necessário que as metas especificas e genéricas impostas pelas áreas de processo, relativas ao nível em que se encontra a empresa, sejam cumpridas.

- Nível 1: Inicial

  Os processos neste nível são desorganizados e caóticos, não existe qualquer tipo de plataforma a seguir. O sucesso é muitas das vezes conseguido através de ações individuais, e não devido a um plano bem delineado de tarefas. As empresas no entanto conseguem gerar produtos que funcionam, mas excedem muitas das vezes orçamentos e prazos estipulados.

- Nível 2: Gerido
  Existe uma verificação de requisitos, e os processos passam a ser controlados e planeados. Para evoluir para o nível 3, os objetivos específicos e genéricos propostos pelas áreas de processo que contemplam este nível têm de ser atingidos.
  - *Configuration Management* (CM)
  - *Measurement and Analysis* (MA)
  - *Process and Product Quality Assurance* (PPQA)
  - *Requirements Management* (REQM)
  - *Supplier Agreement Management* (SAM)
  - *Service Delivery* (SD)
  - *Work Monitoring and Control* (WMC)
  - *Work Planning* (WP)
- Nível 3: Definido
  São criadas um conjunto de normas e processos padrão a ser cumpridos a nível interno na empresa. Estes descrevem procedimentos, métodos e ferramentas que devem ser adotados.
  - *Capacity and Availability Management* (CAM)
  - *Decision Analysis and Resolution* (DAR)
  - *Incident Resolution and Prevention* (IRP)
  - *Integrated Work Management* (IWM)
  - *Organizational Process Definition* (OPD)
  - *Organizational Process Focus* (OPF)
  - *Organizational Training* (OT)
  - *Risk Management* (RSKM)
  - *Service Continuity* (SCON)
  - *Service System Development* (SSD)
  - *Service System Transition* (SST)
  - *Strategic Service Management* (STSM)
- Nível 4: Gerido Quantitativamente
  São selecionados dados que contribuem para um aumento do desempenho e da qualidade do processo, dados esses que são posteriormente analisados estatisticamente e recorrendo a técnicas quantitativas. A informação recolhida e analisada é armazenada para ser utilizada no futuro, deste modo, torna o processo quantitativamente previsível. Ao contrário do que se passa no nível 3, em que os processos são previsíveis mas de forma qualitativa.
  - *Organizational Process Performance* (OPP)
  - *Quantitative Project Management* (QWM)

- Nível 5: Em otimização
  São estabelecidos objetivos de melhoria de processo, a empresa aposta numa melhoria com base na análise quantitativa feita anteriormente. Esses objetivos são continuadamente revistos, para que possam acompanhar as mudanças de negócio e utilizados como critério na gestão de melhoria de processos.
  - *Causal Analysis and Resolution* (CAR)
  - *Organizational Process Management* (OPM)

2.2.3 *Causal Analysis and Resolution*

O objetivo da *Causal Analysis and Resolution* (CAR), é numa pequena amostra de resultados identificar causas que estejam a pôr em causa o desempenho do processo, e tomar medidas para o melhorar. A CAR ajuda a aumentar a qualidade e a produtividade, impedindo a introdução de defeitos ou outros problemas e identificando as causas que os provocam. Situa-se no nível 5 de maturidade, e funciona como prevenção de defeitos no futuro. A CAR é composto por duas atividades, identificar e analisar, e agir [27]:

– Identificar e analisar as causas dos dados selecionados, dados estes que representam defeitos e problemas que podem ser prevenidos no futuro, e ações de sucesso que possam ser implementadas na empresa.

– Ações que devem ser executadas:

  Remover as causas e prevenir a recorrência dos defeitos já identificados.

  Identificar potenciais defeitos que ainda podem vir a ocorrer.

  Aplicar os fatores de sucesso no processo, para que aumente o seu desempenho.

As causas são encontradas, analisadas e depois corrigidas ou eliminadas; desta forma, os defeitos provocados por essas causas deixam de ser repetidos, aumentando a produtividade e a qualidade dos processos e reduzindo o custo da qualidade. Torna-se mais eficaz, a nível de orçamento de projeto, introduzir a CAR em cada fase do processo de execução do produto, apostando na prevenção dos defeitos e não na sua tardia correção. Os dados analisados podem ser similares em outras áreas ou fases do projeto a ser executado, assim sendo, a CAR funciona também como um meio de aprendizagem entre grupos de trabalho. As atividades do CAR proporcionam mecanismos para que os grupos de trabalho possam avaliar os seus processos, e procurar métodos de melhoria a serem implementados. Quando esse método prova ser eficaz, este é documentado para que seja usado no futuro a nível global na empresa [27].

Tal como foi explicado na secção 2.2.1, uma área de processo (**??**) é constituída por objetivos específicos (*Specific Goals* - SG) e genéricos (*Generic Goals*). Um objetivo específico é composto por práticas específicas (*Specific Practice* - SP), estes objetivos e práticas têm de ser cumpridos e seguidos para que a área de processo, onde estão inseridos, seja aprovada [27].

Na figura 2.14 está ilustrada a área de processo *Causal Analysis and Resolution*; é uma área de processo constituída por dois objetivos específicos e por cinco práticas específicas.

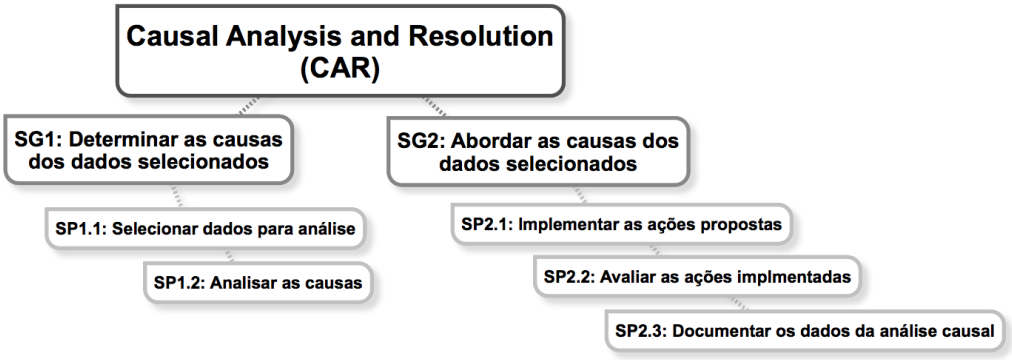

Figura 2.14: Objetivos e práticas específicas CAR

SG 1 Determinar as causas dos dados selecionados

São encontradas as causas sistemáticas fontes de defeitos e outros problemas. Quando a causa é eliminada, o impacto do respetivo defeito pode ser reduzido ou até mesmo eliminado [25].

- SP 1.1 Selecionar os dados para análise: esta atividade pode ser desencadeada ou planeada de forma periódica, no início de casa nova fase ou tarefa.
  - Sub-prática 1: Reunir dados relevantes
  - Sub-prática 2: Determinar quais os dados a analisar
  - Sub-prática 3: Definir formalmente a finalidade da análise, incluindo uma definição clara da melhoria necessária ou esperada.
- SP 1.2 Analisar as causas: o objetivo desta atividade é definir um um conjunto de ações de como abordar os dados recolhidos, e estabelecer um plano de ação de implementação.
  - Sub-prática 1: Conduzir a análise causal em conjunto com as pessoas envolvidas no projeto.
  - Sub-prática 2: Analisar os resultados selecionados para determinar as suas causas.
  - Sub-prática 3: Agrupar os resultados selecionados com base nas causas encontradas.
  - Sub-prática 4: Criar e documentar propostas de ação a serem tomadas para prevenir a ocorrência futura das causas encontradas.

SG 2 Abordar as causas dos dados selecionados

As causas que provocam os defeitos e outros problemas são abordadas de forma sistemática, para prevenir que ocorram novamente no futuro. Grupos de trabalho que operam de acordo com processos bem definidos, analisam se são necessárias melhorias e implementam mudanças de processo para resolver as causas dos dados selecionados.

- SP 2.1 Implementar as ações propostas: pôr em prática as propostas de ação que foram desenvolvidas na análise causal. Este plano descreve tarefas a serem realizadas para eliminar as causas dos defeitos ou dos problemas analisados para que estes não sejam repetidos [25].
  - Sub-prática 1: Analisar as propostas de ação e determinar as suas prioridades.
  - Sub-prática 2: Selecionar as propostas de ação a serem implementadas.
  - Sub-prática 3: Criar um plano de ação para implementar as ações selecionadas.
  - Sub-prática 4: Implementar o plano de ação.
- SP 2.2 Avaliar as ações implementadas: depois da implementação do plano de ação, é necessário uma avaliação para verificar se o desempenho do processo foi ou não melhorado [25].
  - Sub-prática 1: Medir e analisar as mudanças no desempenho do processo.
  - Sub-prática 2: Determinar o impacto da mudança nos objetivos de desempenho da qualidade.
  - Sub-prática 3: Determinar e documentar as ações que não obtiveram os resultados esperados.
- SP 2.3 Documentar os dados da análise causal: registar os dados obtidos da análise causal, assim como todo os dados de resolução de defeitos e problemas, para que possam ser utilizados por toda a organização [25].
  - Sub-prática 1: Tornar os dados disponíveis para que outros grupos de trabalhos sejam capazes de mudar os processos com o objetivo de alcançar resultados semelhantes.
  - Sub-prática 2: Apresentar as propostas de melhoria para toda a organização quando as ações implementadas mostraram ser eficazes para o grupo de trabalho.

2.3 Análise Causal de Defeitos

Gerir e aprender com os "erros"cometidos é bastante importante para quem desenvolve *software*. Este subcapítulo apresenta uma visão geral das técnicas mais utilizadas na análise de defeitos. Na visão de Card, um defeito é um segmento de *software* ou documentação que precisa de ser alterada, de modo a satisfazer os critérios do projeto [28]. A analise causal de defeitos, *(Defect Causal Analysis* (DCA), pretende aplicar a análise causal e resolução (CAR) a um tipo específico de problema. Trata-se de um um plano de prevenção de defeitos, que também é responsável pela implementação das ações corretivas [29].

O QUE É A ANÁLISE CAUSAL

Análise causal (AC) é constituída por um sistema causal, sistema este que consiste num conjunto interligado de eventos e condições que originam consequências. A AC é a investigação desse sistema para identificar ações que o influenciem, com o objetivo de minimizar as consequências.

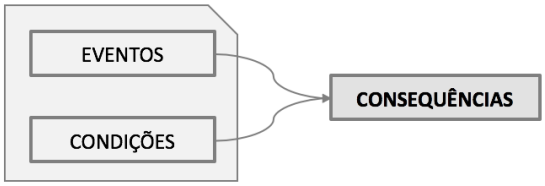

Figura 2.15: Sistema causal

Existem três condições que têm de ser estabelecidas na AC [28]:

- Tem de existir associação entre a causa e o efeito;
- Causa precede o efeito;
- O elo de ligação entre os dois tem de ser identificado.

Figura 2.16: Condições AC

PRINCÍPIOS DA DCA

Existem três princípios comuns para conduzir abordagens baseadas na DCA, para uma melhoria da qualidade [28]:

- Reduzir e prevenir os defeitos para melhorar a qualidade: onde existem muitos defeitos existe também falta de qualidade, seja qual for a definição que lhe seja dada. No entanto, podemos melhorar a qualidade de software apostando na prevenção e numa deteção prévia de defeitos.

- Conhecimento interno: análise causal pode ser conduzida por pessoas externas, no entanto não conhecem o produto com tanto pormenor como as pessoas que o fabricaram. Daí ser necessário uma ajuda interna, alguém que conheça o produto/software e que tenha estado presente no momento em que os defeitos foram introduzidos.

- Focar nos erros sistemáticos: erros sistemáticos são erros que aparecem devido à introdução de defeitos similares, e podem aparecer em diferentes etapas do projeto. A DCA ajuda a identificar e a prevenir esses erros.

2.3.1 Elementos de um sistema causal

Para facilitar uma melhor compreensão sobre os sistemas causais e o seu planeamento, Card [28], criou um modelo e uma terminologia para um sistema causal. A figura 2.17 ilustra os vários elementos que constituem um sistema causal e a forma como estão organizados.

Um sistema causal é composto por três classes de elementos:

- Observações: os eventos e condições que englobam o sistema causal.
- Ações: os esforços para influenciar o comportamento do sistema causal.
- Objetivos: finalidade da investigação do sistema causal.

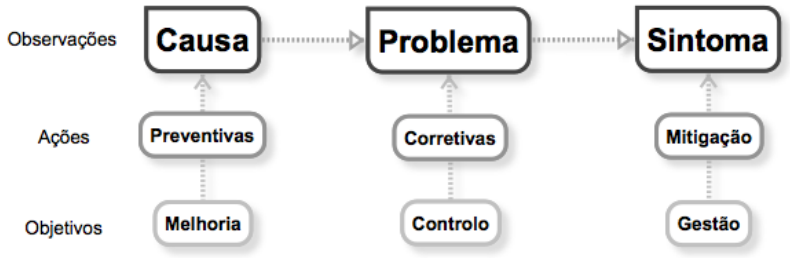

Figura 2.17: Elementos de um sistema causal

Observações são condições e eventos que podem ser detetados, e para isso é necessário descobrir a ligação entre eles:

- Causas: são os eventos e condições que vão contribuir para aparecimento do problema;
- Problema: um problema se corrigido elimina futuros sintomas;
- Sintoma: aparecem no seguimento do problema, que mesmo sendo tratado não faz com que o problema desapareça mas consegue atenuar as consequências.

Depois de um sistema causal ser percebido, podem ser adotadas ações para alterar o seu impacto na organização. As ações podem ser do tipo:

- Preventivas: ações preventivas ajudam a diminuir a probabilidade de problemas similares ocorrerem novamente;
- Corretivas: resolve o problema diretamente;
- Mitigação: combater as consequências adversas dos problemas, contrariam os sintomas do problemas.

A combinação ideal entre as ações (preventivas, corretivas e de mitigação), depende do custo da sua execução e as percussões dos sintomas produzidos. Para usar a análise causal é necessário ter em conta 3 objetivos:

- Melhoria: oportunidade de melhorar e aumentar a performance dos objetivos propostos;
- Controlo: desencadeado por resultados anteriores,
- Gestão: gerir um plano já existente ou estabelecer novas metas a serem atingidas.

A maioria dos sistemas causais tende a ser bastante complexo na sua estrutura. Isto é, um problema específico pode originar diversos sintomas. E ainda, uma causa pode contribuir para o aparecimento de múltiplos problemas. Consequentemente, podem ser tomadas inúmeras ações [28]. Uma boa compreensão dos conceitos básicos e do vocabulário usado ajuda numa melhor implementação da análise causal.

2.3.2 Ferramentas usadas

MODELOS PARA A GESTÃO DA QUALIDADE

Modeling for Quality Management (MQM), tem como objetivo analisar os defeitos encontrados nas diferentes fases, e verificar se estes são ou não consistentes e aceitáveis no plano do projeto. O *Orthogonal Defect Classification* (ODC), é um dos modelos utilizados pela MQM e envolve acompanhar e analisar os vários atributos dos defeitos [28]. ODC é um esquema que apoia na definição e classificação de atributos de defeitos, permitindo deste modo uma análise matemática e modelagem possível. A análise ODC permite um diagnóstico que facilita a avaliação das várias fases da execução de um projeto (conceção, desenvolvimento, teste e serviço) [30]; ODC é compostao por sete atributos: *assignment/initialization, checking, algorithm/method, function/class/object, timing/serialization, interface/O-O messages, relationship* [31]. Estes atributos foram definidos pela IBM e podem aplicar-se a qualquer projeto, independentemente do processo ou produto [32].

- *Assignment/initialization*: valor atribuído incorretamente ou não atribuído de todo:
  - Variável interna não tem valor correcto, ou não tem qualquer valor;
  - Inicialização de parâmetros.
- *Checking*: erros causados por validação de parâmetros ou dados incorretos ou ausentes:
  - valor maior que 100 não é válido, e a verificação para certificar que o valor era inferior a 100 está em falta;
  - O ciclo deveria ter parado na nona iteração, mas o ciclo continuou até à próxima iteração.

- *Algorithm/method*: problemas de eficiência ou de correção que afetam a tarefa e podem ser corrigidos pela implementação de um algoritmo ou estrutura de dados local, sem a necessidade de solicitar uma alteração de design:
  - O número e/ou tipos de parâmetros de um método ou uma operação está especificado incorretamente;
  - Um método ou operação não foi tornada pública na especificação de uma classe.
- *Function/class/object*: o erro exige um alteração da estrutura formal, afetando as interfaces com o usuário final, interfaces do produto, interface de arquitetura de hardware, ou estrutura de dados global.
  - Uma classe C++ foi omitida durante a conceção do sistema;
  - Uma base de dados não inclui um campo de endereço, embora estejam especificados nos requisitos.
- *Timing/serialization*: serialização do recurso compartilhado estava em falta, o recurso errado foi serializado:
  - Faltou a serialização ao fazer alterações a um bloco de controlo compartilhado;
  - Um bloqueio hierárquico está a ser usado, no entanto o código defeituoso não conseguiu adquirir os bloqueios na sequência prescrita.
- *Interface/O-O messages*: problemas de comunicação entre módulos, componentes, objeto, funções via macros, declarações, lista de parâmetros:
  - O número e/ou tipos de parâmetros da *O-O message* não estão conforme o serviço solicitado.
- *Relationship*: problemas relacionados com associações entre procedimentos, estruturas de dados e objetos:
  - A relação de herança entre duas classes está ausente ou incorretamente especificada;
  - O limite para o número de objetos que podem ser instanciados a partir de uma determinada classe está incorreta e provoca a degradação do desempenho do sistema.

CONTROLO DO DESEMPENHO DO PROCESSO

O desempenho do processo pode ser avaliado a ponto de determinar se o processo está ou não bem estruturado. O controlo estatístico de processos (*Statistical Process Control* - SPC) foi desenvolvido para gerir processos [28], é uma metodologia padrão usada para medir e controlar a qualidade durante a execução de um processo. São obtidos dados em tempo real da qualidade através do processo ou do produto. SPC é uma abordagem com técnicas de apoio para resolver problemas. A sua principal função é controlar a variação dos resultados dos processos que são estáveis e previsíveis. Os gráficos de controlo (2.18 a) [33] mostram a variação de uma medição

durante o período de tempo em que o processo é observado; por sua vez gráficos como histogramas (2.18 b) [33] mostram um resumo dos resultados.

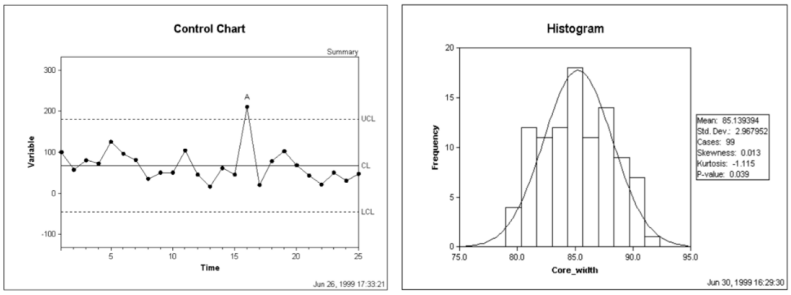

Figura 2.18: Gráfico de controlo (a) Histograma (b)

As fases preparatórias do SPC envolvem várias etapas usando uma série de ferramentas diferentes, ferramentas estas que ajudam as organizações a compreender e a melhorar os seus processos [33]: folha de verificação, causa e efeito, digrama de pareto, digrama de dispersão, histogramas e gráficos de controlo, são alguns dos recursos usados pelo SPC. O SPC está divido em quatro passos [28]:

– Desenvolver um plano de controlo,

– Executar um processo,

– Medir os resultados,

– Usar os resultados obtidos para estabilizar o processo e depois melhora-lo.

A estabilidade é o resultado do controlo, e para que isso aconteça é necessário que o desempenho do processo seja avaliada durante a sua execução e não apenas no fim [28]. Sempre que um processo é recomeçado, são gerados resultados diferentes, mesmo que estejam garantidas as mesmas condições anteriormente estabelecidas.

Essas diferenças podem ser classificadas em dois tipos de causas: causas comuns e causas especiais [34]. As causas comuns são inúmeras fontes de variação que atuam de forma aleatória no processo; estas alterações representam o padrão natural do processo. Um processo que represente causas comuns é considerado um processo estável ou sob controlo. As causas especiais são causas que não são pequenas e não seguem um padrão aleatório, são chamadas também de causas assinaláveis; fazem com que o processo se desvie do seu curso normal. Reduzem também o desempenho do processo, devem ser identificadas e eliminadas [28].

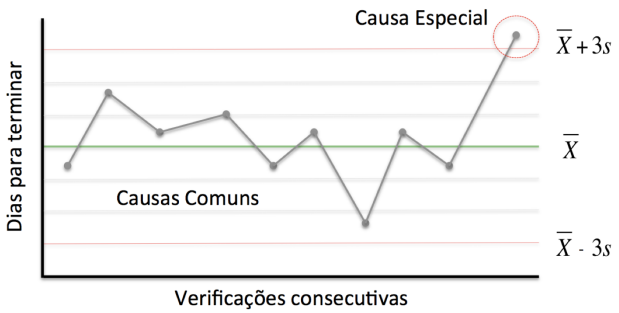

Figura 2.19: Ilustração de causas comuns e causas especiais

A figura 2.19 [28] mostra um gráfico de controlo, e tem como objetivo estudar a variação do processo visualmente. Os pontos que surgem no gráfico são subgrupos, e cada subgrupo pode conter múltiplas observações [28]. Os dados são convertidos para o gráfico com limites de controlo pré-determinados; os limites de controlo são determinados pela capacidade do processo, enquanto que os limites de especificação são determinados pelas necessidades do cliente [35].

- Gráfico de Pareto

  O diagrama de Pareto pode ser usado para exibir graficamente categorias de problemas para que sejam corretamente priorizados. Além de fornecer um meio para estudar e melhorar a qualidade, fornece também um meio para estudar e melhorar a eficiência, desperdício de material, questões de segurança, redução de custos, etc. [36]. Muitas vezes há aspetos diferentes de um processo que podem ser melhorados, como o número de produtos com defeito, tempo de produção e até economias de custo. Cada aspeto contém pequenos problemas, o que torna difícil de os abordar. Um gráfico de Pareto indica quais os problemas a atacar primeiro, mostrando a proporção do problema total que cada um dos problemas pequenos compreendem. Esta regra baseia-se no principio de Pareto: 20% das causas provocam 80 % dos problemas [36]. Em suma um gráfico de Pareto ajuda a agrupar os defeitos de acordo com a sua classificação. evidenciando as categorias mais comuns e a soma delas [37].

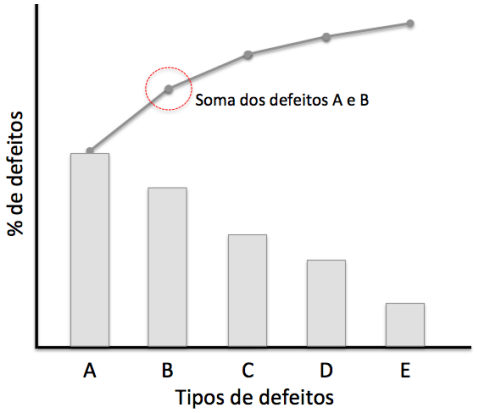

Figura 2.20: Gráfico de Pareto

- Diagrama de causa-efeito ou espinha de peixe (*Fishbone Diagram*) .
  Uma ferramenta de análise é o diagrama de causa-efeito (CE) ou *Fishbone* (diagrama de espinha de peixe), também conhecido como diagrama de *Ishikawa*, devido a Karou Ishikawa que o desenvolveu em 1943 [34]. Este diagrama organiza e mostra as relações entre as diferentes causas para o problema que está a ser explorado. As principais causas são conectadas diretamente ao ramo principal da categoria, e as várias causas secundárias estão ligados aos ramos das causas principais [36]. Assim, um diagrama CE auxilia o grupo de trabalho a [34]:

  – Reunir e organizar as possíveis causas para aquele problema,
  – Chegar a um acordo sobre o problema,
  – Expor as lacunas de conhecimento existentes,
  – Classificar as causas mais prováveis,
  – Estudar cada causa.

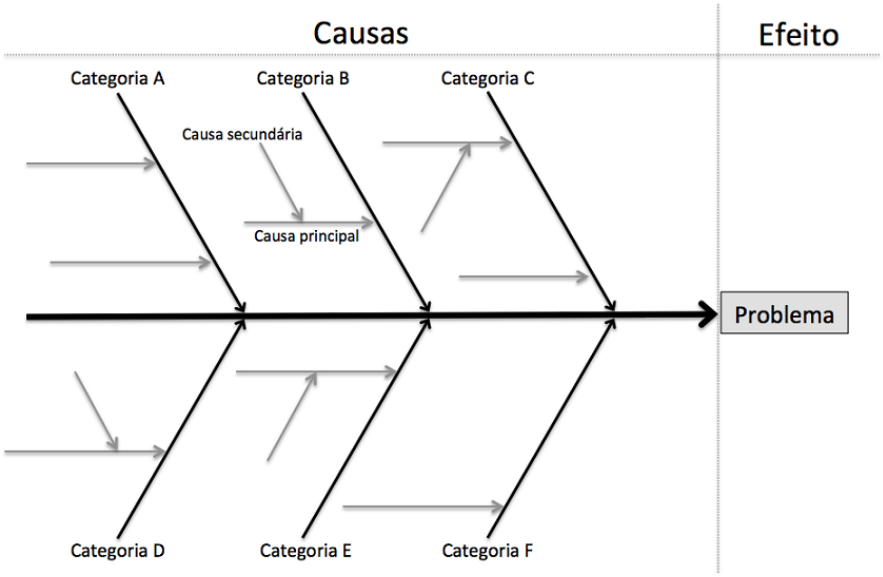

Figura 2.21: Diagrama de causa-efeito (*Fishbone Diagram* ou de *Ishikawa*)

2.3.3 Procedimentos da DCA

David. N. Card [38], resume a Análise Causal (AC) em seis passos: (1) selecionar uma amostra de defeitos, (2) classificar os defeitos selecionados, (3) identificar erros sistemáticos, (4) identificar as principais causas, (5) desenvolver propostas de ação, (6) documentar os resultados da reunião. Neste contexto, um erro sistemático é um erro que resulta em defeitos semelhantes que se vão repetindo em diferentes etapas.

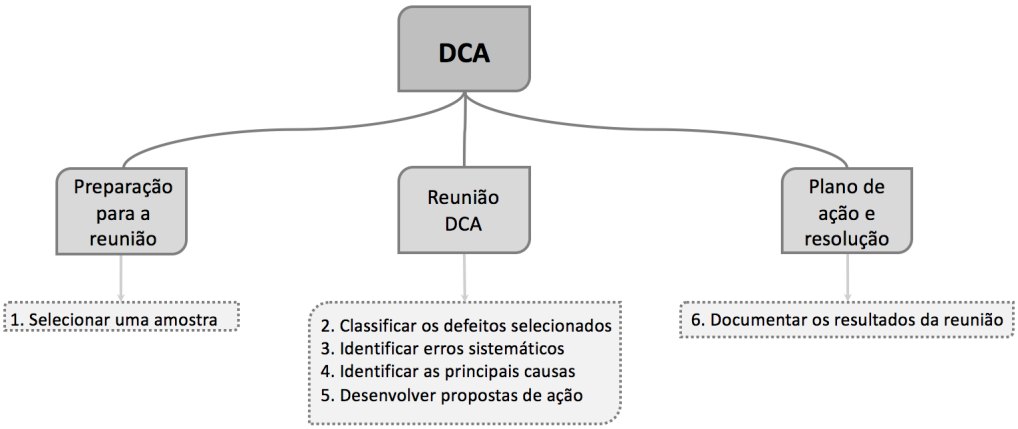

Figura 2.22: Processos e passos da DCA

1. Selecionar uma amostra de defeitos

A amostra selecionada pode surgir de duas formas: análise lógica e análise estatística. Análise lógica é uma análise determinista que examina a ligação lógica entre os efeitos e as causas correspondentes, e estabelece relações causais gerais. A análise estatística é uma análise probabilística que examina a ligação estatística entre causas e efeitos e deduz as relações causais prováveis entre os dois. Esta última é acionada através de uma anomalia no gráfico de controlo estatístico de processos (2.23); são os dados relacionados com essa instabilidade que vão ser analisados [39].

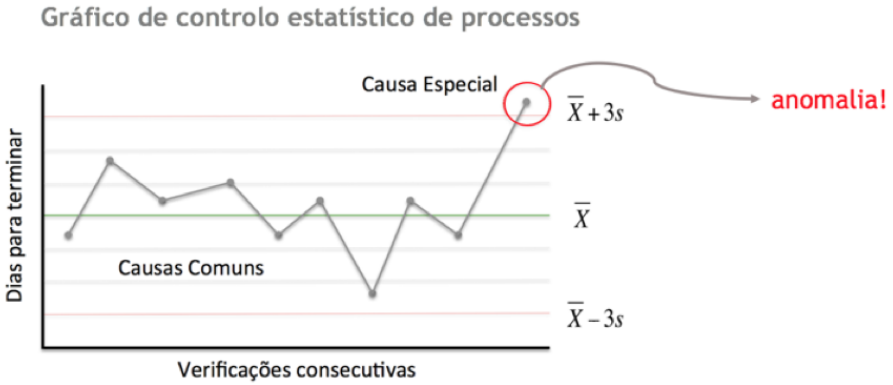

Figura 2.23: Gráfico exemplo de controlo estatístico de processos

2. Classificar os defeitos selecionados

Após serem recolhidos os dados é necessário classificar ou agrupar por categorias. Entre as informações mais comuns encontram-se o tipo de defeito, a fase em que foi inserido, fase em que foi detetado. É usado o *Orthogonal Defect Classification* (ODC) que sugere atributos de modo a classificar os defeitos. No entanto, e consoante a necessidade pode ser criada outro tipo de classificação. Outra ferramenta utilizada nesta etapa são os gráficos de Pareto, que ajuda a agrupar

os defeitos consoante a sua classificação [28]. A figura 2.24 [40] serve de exemplo onde foi usada uma classificação diferente para agrupar os defeitos; mostra também que a maioria dos defeitos são do tipo "*Incorrect Fact*", e que a sua soma com o tipo "*Omission*" representa cerca de 60% de todos os defeitos encontrados.

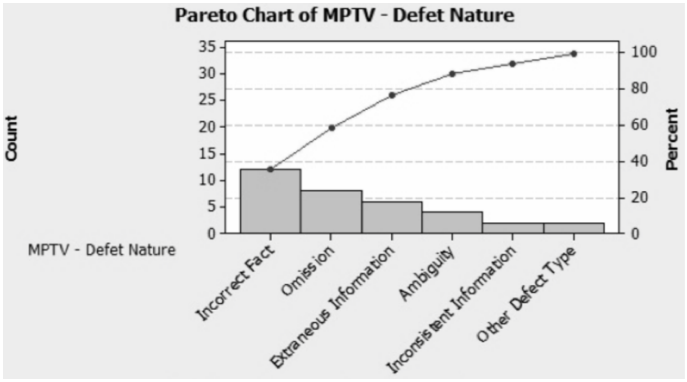

Figura 2.24: Gráfico de Pareto

3. Identificar erros sistemáticos

Como já foi referido anteriormente, um erro sistemático é um erro que aparece devido a defeitos similares e que ocorrem em diferentes alturas do processo. Cerca de 20% a 40% dos defeitos selecionados na amostra inicial são associados a erros sistemáticos[28]. Uma percentagem elevada deste tipo de erros, motiva a uma alteração nos processos organizacionais.

4. Identificar as principais causas

Como não é economicamente viável tratar todos os fatores que podem causar erros sistemáticos, é necessário identificar quais são as principais causas que os originam. Para ajudar nessa tarefa usamos o diagrama causa-efeito ou espinha de peixe. Este diagrama ajuda a explorar de uma maneira mais profunda o problema. Usualmente os defeitos podem "cair"numa das quatro categorias: métodos, equipamento/meio ambiente, pessoas e requisitos. Mais uma vez, essas categorias podem ser alteradas de acordo com a especificidade do defeito.

A figura 2.25 [40] representa uma diagrama causa-efeito para o tipo de defeito *Incorrect Fact*; são apresentadas as principais causas que contribuíram para aquele tipo de defeito.

O diagrama mostra que normalmente as causas para o tipo de defeito classificado como "*Incorrect Facts*"foram "*lack of domain knowledge*"(25%), "*size and complexity of the problem domain*"(18,7%), "*oversight*"(15,6%) e "*limited tool user for traceabillity*"(12,5%).

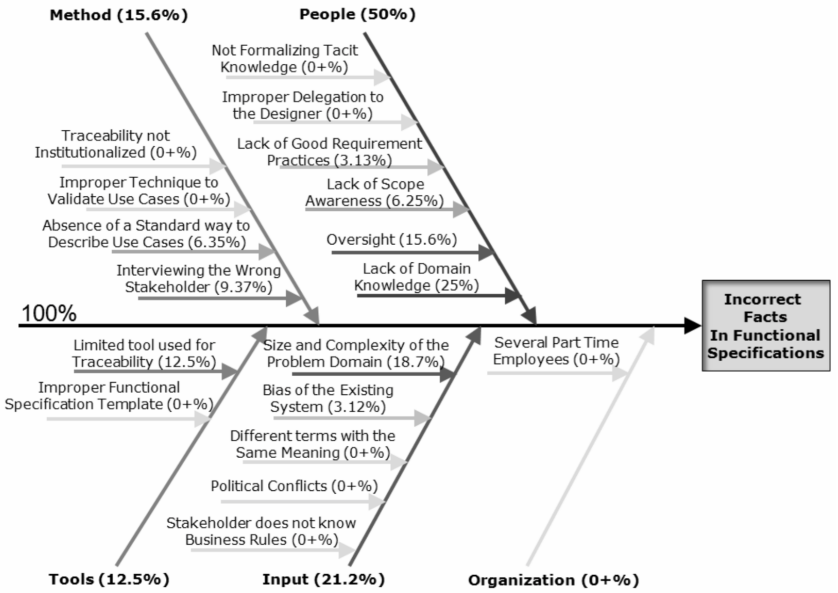

Figura 2.25: Gráfico de causa-efeito

.

5. Desenvolver propostas de ação

Nesta etapa, devem ser identificadas as ações para melhorar os processos de desenvolvimento, a fim de prevenir as causas identificadas. As propostas devem de ser específicas e dirigidas às principais causas do erro sistemático. O número de ações deve ser pequeno, mas eficaz. A DCA destaca-se dos processos mais genéricos por propor ações específicas, em vez de abordar áreas mais vastas, que por sua vez vão carecer de uma maior atenção para melhorar o processo [40]. Exemplo de ações propostas para "*lack of domain knowledge*"e "*size and complexity of the problem domain*", podem ser [40]:

(i) Estudar a área e o sistema pré-existente;

(ii) Modificar o modelo das especificações funcionais, criando uma secção para as regras de negócio.

6. Documentar os resultados da reunião

Para assegurar que as soluções encontradas anteriormente são executadas é essencial que estas sejam documentadas. Card [28], refere que deve ser criada uma equipa para gerir o plano de ação que foi elaborado, e também para garantir que é posto em prática.

Ao prevenir os defeitos é possível melhorar a qualidade, diminuir o tempo de entrega ao cliente, e reduzir o custo do projeto, nomeadamente o *rework*. O "refazer"trabalho já feito resulta de defeitos encontrados, e consome cerca de 30 a 50% do orçamento do projeto

Capítulo 3

Aplicação na Critical Manufacturing

A *Critical Manufacturing* (CM) é formada por uma equipa de analistas e engenheiros, que em conjunto trabalham para oferecer soluções inovadoras que correspondem às necessidades dos seus clientes. A estratégia da CM está assente na indústria de produção discreta, onde oferece soluções de gestão e controlo de produção que capacita as unidades fabris para alcançarem os seus objetivos. O seu produto permite detalhar as operações, assim como facilitar a visibilidade sobre os processos e custos refletidos na cadeia de abastecimento; é também bastante modular permitindo uma fácil implementação em infraestruturas já existentes.

Embora espalhada um pouco por todo mundo, com subsidiárias em Dresden (Alemanha), Suzhou (China), Austin (EUA) e um escritório comercial em Taiwan, a sua sede e centro de engenharia está localizada no Tecmaia Parque de Ciência e Tecnologia da Maia [1].

3.1 Política de qualidade na CM

A CM possui inúmeros prémios e certificações. As duas certificações mais relevantes são a norma ISO:9001 e o nível 3 do CMMI, pelo que o seu sistema de gestão da qualidade foi estabelecido de forma a cumprir com os requisitos destas normas. Como já foi referido, este conjunto de normas especifica os requisitos que a empresa tem de cumprir para demonstrar que é capaz de criar um produto que vá de encontro às necessidades do cliente, aumentando a satisfação do mesmo. Define deste modo os requisitos de documentação, política de qualidade, normas de gestão de recursos, normas para a realização do produto e, por fim, normas para medição e análise e melhoria de processos. Os projetos na CM têm um objetivo bem definido e um equipa diversificada, com papéis claros e bem definidos; as pessoas envolvidas na equipa dependem sempre das necessidades do cliente e das características do projeto. As equipas podem ser constituídas por [41]:

- *Project Manager*
- *Software Development Team*
- *Configuration Manager*
- *Quality Manager*

Durante um projeto existem três pontos que devem ser cumpridos [41]:

- *Kick-off meeting*: esta reunião é da responsabilidade do *Project Manager* e representa o início oficial do projeto; em conjunto com o cliente, são delineados o objetivo, esforço estimado e entregas;

- *Customer acceptance*: este marco representa a confirmação do cliente de que o que foi solicitado foi entregue de acordo com as expectativas; de acordo com o ciclo de vida do projeto, podem existir ou não mais encontros deste tipo;

- *Close-down meeting*: é a ultima etapa do projeto; representa o fim de todas as atividades relacionadas com o projeto, embora as atividades de suporte ou de manutenção possam ainda ocorrer após o encerramento do projeto. A documentação é finalizada, e as ações aprendidas durante o desenvolvimento do projeto são apresentadas para melhoria de processos.

Na CM, são várias as atividades executadas para garantir que o cliente recebe o produto esperado. As atividades que são desenvolvidas durante o desenvolvimento de um projeto vão contribuir para o sucesso do mesmo; são atividades que podem ser realizadas com mais ou menos formalidade e detalhe, executadas em modo paralelo ou sequencial; todo o plano de atividades depende sempre do ciclo de vida do projeto (figura 3.1 [41]) e do que foi acordado com o cliente. Essas atividades podem ser [41]:

- *Requirements Analysis*: refere-se à recolha, compreensão, registo e análise das necessidades dos clientes (requisitos) a serem implementados na solução;

- *Design*: refere-se à compreensão da equipa alocada ao projeto de como o software pode ser modelado/projetado para acomodar os requisitos;

- *Development*: diz respeito à codificação e aos testes unitários dos requisitos de acordo com o *design*;

- *Verification*: compreende os testes e/ou revisão de código para verificar se o projeto está de acordo com os requisitos solicitados;

- *Release*: refere-se à preparação para distribuição de todos os componentes do software e de toda a documentação a ele relacionada;

- *Support*: diz respeito ao apoio ao cliente depois do lançamento do produto, de modo a esclarecer dúvidas e corrigir quaisquer problemas.

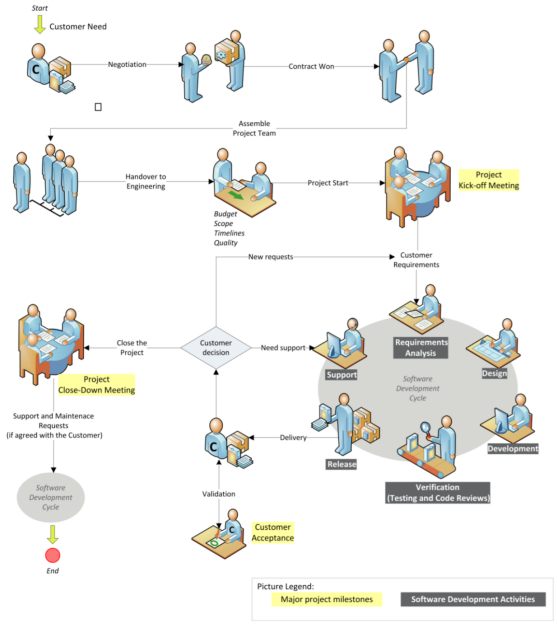

Figura 3.1: Ciclo de projeto da CM

Na CM existem duas abordagens possíveis como metodologias de desenvolvimento, métodos ágeis (*agile*) ou cascata (*waterfall*) [41]:

- *Agile* (figura 3.2 [41]): entregas em ciclo curtos, realizadas a cada 2 a 4 semanas, onde um conjunto de requisitos definidos é implementado e validado pelo cliente.

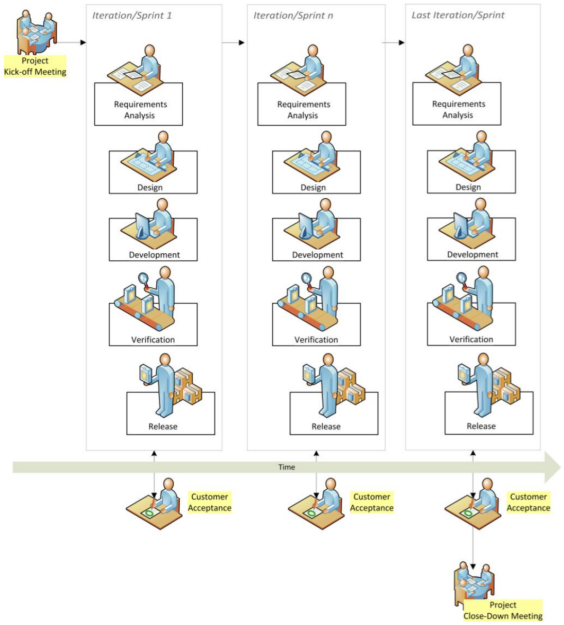

Figura 3.2: Ciclo de Desenvolvimento Baseado nos Métodos Ágeis

- *Waterfall* (fig.3.3 [41]): a entrega ao cliente é feita depois de transitar de uma fase para outra.

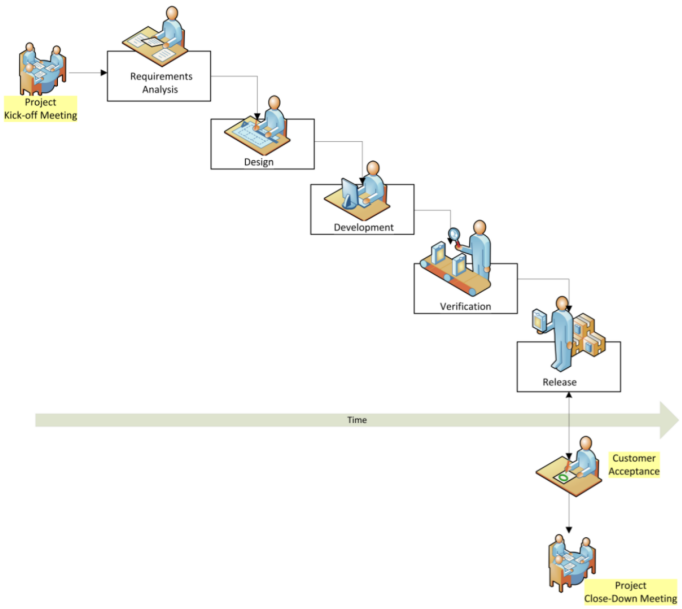

Figura 3.3: Modelo Cascata

O cliente está mais envolvido na abordagem *agile*, os ciclos de validação são mais curtos, bem como o *feedback*. Este tipo de abordagem minimiza o risco durante o projeto, permite a possibilidade de um *feedback* constante, afim de se adaptar rapidamente a novos pedidos ou alterações. Esta abordagem é a mais utilizada pela *Critical Manufacturing* [42].

O Produto

O cmNAVIGO MES (*Manufacturing Execution System*) é um plataforma tecnológica que oferece às indústrias complexas uma maior agilidade, visibilidade e confiabilidade [43]. Foi especificamente projetado para resolver as dificuldades e deficiências das soluções existentes nas industrias de semicondutores, solar e eletrónica. O cmNavigo proporciona também uma ferramenta de integração e produtividade, uma interface gráfica com o utilizador e uma plataforma de análise estatística inteligente como se poder ver na figura 3.4 [44].

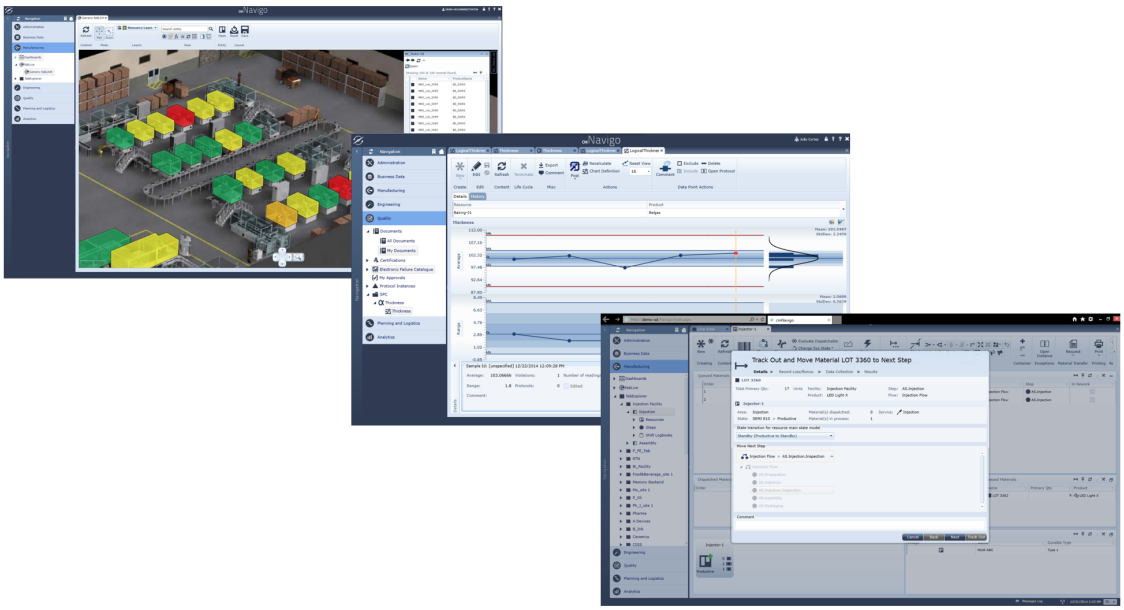

Figura 3.4: cmNAVIGO MES

3.2 DCA na CM

3.2.1 Selecionar uma amostra

A amostra inicial contava com 113 defeitos de diversas categorias (*Build, Customer, Demo, Development* e *Implementation and Modeling*) do produto cmNAVIGO; estas categorias servem apenas para informar a origem do defeito e não uma classificação do defeito em si. Ao registar um defeito no TFS (Team Foundation Server) é necessário o preenchimento de campos informativos, como por exemplo [45]:

- *Story Points*: serve para estimar a quantidade de trabalho necessário para concluir o defeito reportado, usando qualquer unidade de tempo que a equipa preferir;
- *Steps to Reproduce*: capturar informações suficientes para que outros membros da equipa possam compreender o impacto do defeito, bem como os passos para o reproduzir;
- *Priority*: classificação baseada na importância e a urgência de o resolver;
  - 1: o produto não pode ser enviado sem a resolução do defeito;
  - 2: o produto não pode ser enviado sem a resolução do defeito, mas não há urgência na sua resolução;
  - 3: a resolução do defeito é opcional, baseada nos recursos, tempo e risco.
- *Severity*: impacto que o defeito tem no projeto (*Critical, High, Medium* e *Low*) ;

- *Area Path*: área a que o defeito está associado, produto ou equipa;
- *Iteration*: *sprint* ou iteração em que a tarefa vai estar concluída;
- *Bug Type*: em que fase foi encontrado ou reportado o defeito (*Build, Customer, Demo, Development* e *Implementation and Modeling*);

O TFS é usado para gerir todo o ciclo dos projetos, nomeadamente para gerir código, realização de relatórios, gestão de requisitos, gestão de projetos (tanto em *agile* como em *waterfall*), testes, controlo de *release*, abrange todo o ciclo de vida de um projeto [46].

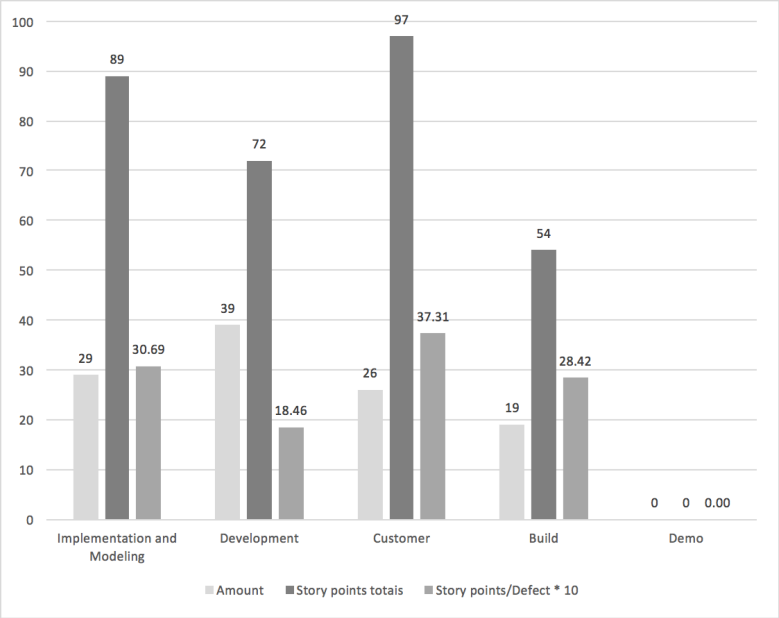

Figura 3.5: Gráfico da análise da amostra inicial

Foram selecionados para estudo todos os defeitos reportados pelo cliente (*Customer*), por serem aqueles que mais *story points* tinham por defeito, e por terem sido detetados pelo utilizador final. Esta amostra final conta com 26 defeitos para análise.

3.2.2 Classificar os defeitos selecionados

Como foi referido na secção 2, o *Orthogonal Defect Classification* (ODC) é apenas um guia de como podem ser classificados os defeitos, no entanto e consoante a necessidade, podem ser criadas novas categorias que se enquadrem melhor no projeto. Para a *Critical Manufacturing* foi necessário criar uma nova classificação:

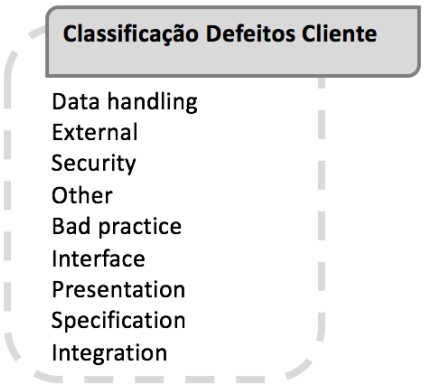

Figura 3.6: Nova classificação

- *Data Handling*: problemas no acesso a dados;

- *External* : causado por software externo à CM; o defeito está no software ou pacote usado pela CM e não no produto em si;

- *Security*: problema a nível de segurança que introduz vulnerabilidades de segurança que podem ser exploradas para obter acesso não autorizado do sistema;

- *Other*: não se enquadram em nenhuma da classificação existente devido à sua descrição ser bastante genérica;

- *Bad Practice*: membro da equipa executou uma má prática durante o desenvolvimento do produto;

- *Interface*: problema de comunicação entre módulos, componentes ou objetos;

- *Presentation*: problema de apresentação gráfica;

- *Specification*: requisitos mal compreendidos ou mal especificados;

- *Integration*: problema de integração de módulos do sistema;

Depois de feita a classificação, os defeitos podem então ser agrupados. Na figura 3.7, encontram-se os defeitos da amostra final, já classificados e agrupados de acordo com a nova classificação. Entretanto, durante a classificação foram encontrados dois defeitos que não correspondiam à característica de defeito, assim sendo, foram identificados como "*not bug*"; foram mal registados no TSF. Para este passo foi necessário um conhecimento interno sobre o produto, assim sendo contamos com a ajuda de dois elementos da equipa de desenvolvimento da CM, o *project manager* e o *scrum master*. Estes dois membros da equipa participaram nos restantes passos da ACD, validando e sugerindo ações que poderiam ser mais tarde implementadas.

Defect Type	Amount	Cumulative	Cumulative %
Data Handling	7	7	29%
Specification	6	13	54%
Presentation	3	16	67%
External	2	18	75%
Other	2	20	83%
Security	1	21	88%
Integration	1	22	92%
Interface	1	23	96%
Bad practice	1	24	100%
Not a Bug	2		
Total	**24**		

Figura 3.7: Classificação dos defeitos

Gráficos de Pareto ou tabelas podem ser produzidas para ajudar a agrupar e a identificar conjuntos de defeitos mais comuns, e apurar onde é mais provável encontrar erros sistemáticos. Analisando o gráfico de Pareto da figura 3.8 , a soma dos defeitos *Data handling*, *Specification* e *Presentation* representa 67% dos defeitos encontrados.

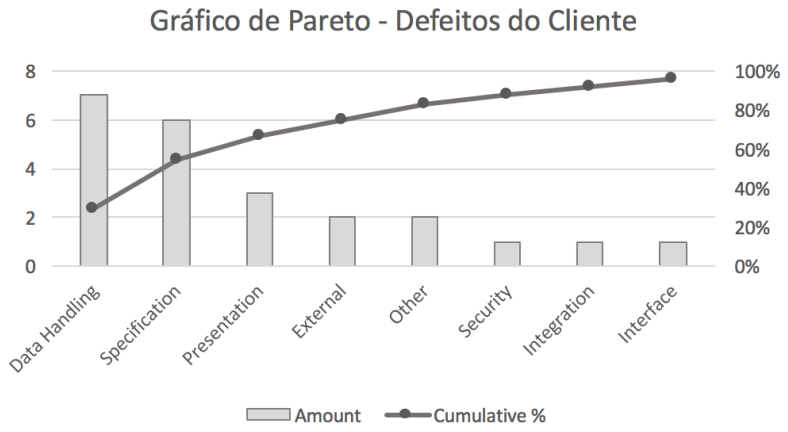

Figura 3.8: Gráfico de Pareto

3.2.3 Identificar erros sistemáticos

Neste passo foram analisados os defeitos classificados como *Data handling*, *Specification* e *Presentation*, por representarem uma maior percentagem da amostra de defeitos (67%). Esta tarefa consiste em analisar os defeitos um a um, de modo a encontrar os erros sistemáticos que lhe deram origem. Apenas os defeitos relacionados com esses erros sistemáticos vão ser considerados no passo quatro da DCA.

#	ID	Defect Category	Error	Systematic Errors
1	5627	Data handling	Inadequate architecture	
2	5286	Data handling	Validation failure (no data)	
3	5625	Data handling	Update incorrect data	
4	5728	Data handling	Update incorrect data	
5	5614	Data handling	Incorrect algorithm	Incorrect algorithm
6	5868	Data handling	Incorrect algorithm	
7	5888	Data handling	Incorrect algorithm	
8	5894	Specification	Incorrect specification	Incorrect specification
9	5636	Specification	Incorrect specification	
10	5715	Specification	Incorrect specification	
11	6013	Specification	Requirement misunderstanding	
12	5724	Specification	Validation Specification missing	
13	5746	Specification	Incomplete specification	
14	5500	Presentation	Incorrect algorithm	Incorrect algorithm
15	5942	Presentation	Incorrect algorithm	
16	5735	Presentation	Incorrect algorithm	

Figura 3.9: Tabela de erros sistemáticos da CM

Foram encontrados três erros sistemáticos nesta análise:
- *Incorrect algorithm* da categoria *Data handling*;
- *Incorrect specification* da categoria *Specification*;
- *Incorrect algorithm* da categoria *Presentation*;

3.2.4 Identificar as principais causas

Não é economicamente viável tratar todos os fatores que podem causar erros sistemáticos. Desta forma a melhor solução é eliminar as causas que os provocam. Uma das técnicas que ajuda a identificar e agrupar as principais causas é a aplicação do diagrama de Ishikawa, ou *Fish Bone Diagram*.

Assim sendo, as causas principais dos erros sistemáticos *Incorrect algorithm* (*Data handling*) e *Incorrect specification* (*Specification*), vão ser da categoria *People* e a sua principal causa é *Oversight*.

Defect Category	Error	Systematic Errors	Category	Cause
Data handling	Incorrect algorithm	Incorrect algorithm	People	Oversight
Data handling	Incorrect algorithm			
Data handling	Incorrect algorithm			
Specification	Incorrect specification	Incorrect specification	People	Oversight
Specification	Incorrect specification			
Specification	Incorrect specification			

Figura 3.10: Categoria e causa principal dos erros sistemáticos

No caso do erro sistemático *Incorrect algorithm* da classificação *Presentation*, vai pertencer igualmente à categoria *People*, sendo a sua causa principal *Lack of know-how*.

Defect Category	Error	Systematic Errors	Category	Cause
Presentation	Incorrect algorithm	Incorrect algorithm	People	Lack of know-how
Presentation	Incorrect algorithm			
Presentation	Incorrect algorithm			

Figura 3.11: Categoria e causa principal dos erros sistemáticos

3.2.5 Desenvolver propostas de ação

Identificadas as principais causas e agrupadas por categorias, é necessário estruturar um plano de ação com propostas a serem implementadas de modo a evitar a sua ocorrência futura. As propostas devem ser específicas e dirigidas às principais causas do erro sistemático. Card [28] menciona ainda que o número de ações deve ser pequeno mas eficaz. A DCA destaca-se dos processos mais genéricos por propor ações específicas, em vez de abordar áreas mais vastas, que por sua vez vão carecer de uma maior atenção na melhoria do processo.

Durante a reunião DCA na CM, e em conjunto com dois elementos de uma das equipas de desenvolvimento do produto cmNavigo, foram apontadas algumas ações, como se pode verificar na figura 3.12, que devem ter em consideração em projetos futuros.

#	ID	Defect Category	Error	Systematic Errors	Category	Cause	Action
5	5614	Data handling	Incorrect algorithm	Incorrect algorithm	People	Oversight	More time in planning to discuss the design
6	5868	Data handling	Incorrect algorithm				
7	5888	Data handling	Incorrect algorithm				Create design task
8	5894	Specification	Incorrect specification	Incorrect specification	People	Oversight	Verify that all requirements are discussed in grooming meeting
9	5636	Specification	Incorrect specification				
10	5715	Specification	Incorrect specification				
14	5500	Presentation	Incorrect algorithm	Incorrect algorithm	People	Lack of know-how	Pair programming (for knowledge sharing)
15	5942	Presentation	Incorrect algorithm				
16	5735	Presentation	Incorrect algorithm				Code review

Figura 3.12: Propostas de ação

De forma complementar, foram estudadas e encontradas ações de melhoria a pôr em prática de modo a prevenir os defeitos analisados. Estas ações estão representadas na figura 3.13.

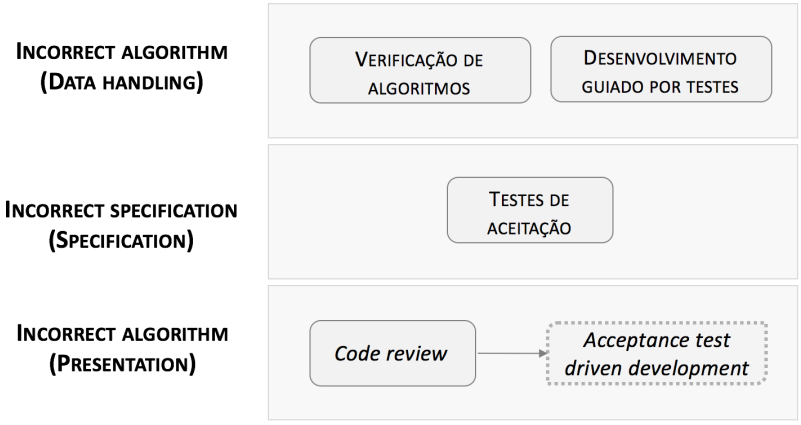

Figura 3.13: Ações de melhoria

VERIFICAÇÃO DE ALGORITMOS

A verificação do algoritmo é um processo no qual se verifica, utilizando técnicas específicas, como tabelas de execução e especificações formais, se o algoritmo fornece os resultados corretos considerando todas as entradas de dados possíveis. Esta verificação é realizada independentemente do código de programação utilizado na implementação. Depois da fase de implementação, o código deverá ser validado, e para que isso ocorra é necessário verificar se o programa está a ser executado corretamente como foi determinado no algoritmo.

DESENVOLVIMENTO GUIADO POR TESTES

Test Driven Development (TDD) é uma metodologia que utiliza testes ao longo do desenvolvimento de software que se baseia em ciclos curtos. As suas fases estão sumariamente enunciadas na figura 3.14.

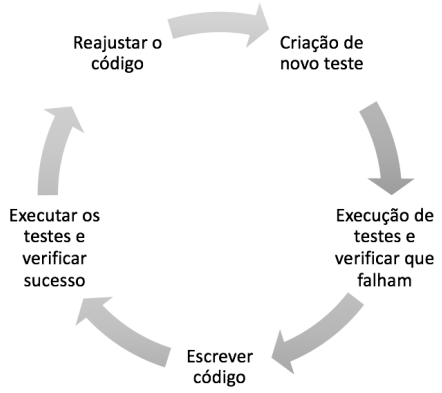

Figura 3.14: Fases do TDD

Como se trata de uma metodologia orientada para testes, estes são escritos em primeiro lugar; inicialmente o *Developer* escreve um caso de teste automatizado que define uma melhoria ou nova

funcionalidade. Este tipo de abordagem permite que o código sejas apenas desenvolvido caso algum teste esteja a falhar e que as duplicações sejam eliminadas. Este tipo de técnica origina que os defeitos sejam encontrados mais cedo no processo de desenvolvimento, reduzindo custo da sua correção [47][48].

TESTES DE ACEITAÇÃO

O principal objetivo deste tipo de teste é confirmar que a aplicação se comporta como estava especificado nos requisitos, representam o interesse do cliente e as suas necessidades. Por outras palavras, permite transmitir confiança que a aplicação possui as características requeridas e se comporta de acordo como o planeado. Os teste de aceitação ajudam a equipa de desenvolvimento em três tarefas [49]:

- Capturam a necessidade dos utilizadores de modo a verificar se o sistema satisfaz essas necessidades;
- Expõe problemas que os testes unitários deixaram passar;
- Fornecem um definição de como o sistema é feito.

Este tipo de teste é considerado completo quando o sistema for aceite pelo utilizador final e a documentação de testes e certificados está concluída [49].

É conveniente que os testes de aceitação sejam definidos junto com as especificações iniciais (*acceptance test driven development*), para ajudar a clarificar as especificações e detetar problemas nas especificações.

CODE REVIEW

Code review, é uma verificação sistemática ao código com o intuito de encontrar erros logo na fase inicial de desenvolvimento, melhorando a qualidade do software. Existem várias formas de *code review*, tais como, *pair programming*, *over-the-shoulder*, *tool-assisted* [50]. No caso da CM, foi sugerido pelos elementos da equipa o *Pair programming*. Este tipo de técnica consiste em dois programadores trabalharem de forma colaborativa no mesmo algoritmo, design ou numa tarefa de programação, lado a lado e num computador. Esta prática melhora a qualidade do design, reduz os defeitos, melhora as habilidades técnicas, e a comunicação em equipa é considerada mais agradável [51].

3.2.6 Documentar os resultados da reunião

Finalmente, devem de ser documentados os resultados da reunião. E este registo é necessário para assegurar que as ações sejam executadas. É aconselhável a criação de uma equipa dedicada a gerir todo o processo de implementação, desde o inicio até à sua conclusão.

Capítulo 4

Aplicação no SIGARRA

A Universidade do Porto (UP) é uma instituição de ensino superior pública, com sede na cidade do Porto, e é atualmente a segunda maior universidade portuguesa com quase 30.000 estudantes distribuídos por 14 faculdades [52][53]. A UP, e todas as suas unidades orgânicas (UO) e organismos, beneficia da utilização de um sistema de informação académico que auxilia as várias atividades de administração e gestão, de ensino, de investigação e desenvolvimento, e de extensão universitária.

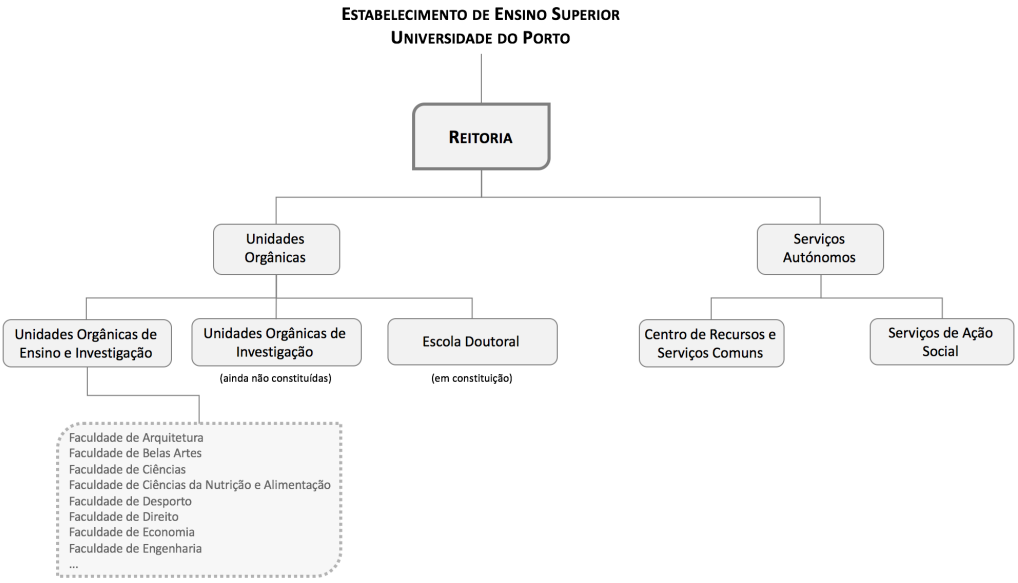

Figura 4.1: Estrutura da Universidade do Porto

Sendo a Universidade do Porto uma instituição de grande envergadura e complexidade, necessitava de uma ferramenta capaz de sustentar todo o seu sistema de gestão de informação, dessa necessidade surgiu o SIGARRA (Sistema de Informação para a Gestão Agregada dos Recursos e dos Registos Académicos). Atualmente, o desenvolvimento e a manutenção do sistema SIGARRA são assegurados por uma equipa técnica conjunta da Reitoria (Universidade Digital), Faculdade de Engenharia e Faculdade de Ciências [54].

4.1 Política de qualidade no Projeto SIGARRA

A U.Porto possui uma dimensão e caraterísticas organizacionais, que necessitam de um sistema de informação abrangente e eficiente, um sistema que suporte as diferentes vertentes da universidade, sendo utilizado por todas as unidades orgânicas. Assim sendo, é necessário para a UP assegurar o bom funcionamento e uma boa utilização por todas as entidades que exploram o SIGARRA. Para que isso seja possível, a gestão do projeto tem de ser eficaz e eficiente, para que consiga responder às necessidades e expectativas da comunidade académica, mantendo o nível de satisfação e de qualidade do SIGARRA elevado. Permitindo deste modo, a manutenção e adaptação dos seus módulos e o desenvolvimento de novas componentes [55].

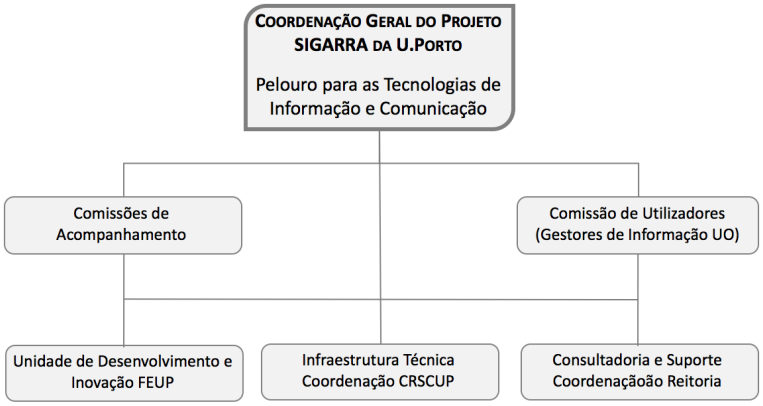

Figura 4.2: Organograma do projeto SIGARRA

A Coordenação Geral do Projeto SIGARRA da U.Porto está entregue à Reitoria da U.Porto, através do Pelouro para as Tecnologias de Informação e Comunicação. São responsáveis por aprovar as orientação e o planeamento das atividades relacionados com os sistemas de informação da UP; e por assegurar a gestão da qualidade do projeto, de forma a garantir que está a ser implementado segundo as melhores normas e processos de desenvolvimento [55].

- Comissões de Acompanhamento: aconselham sobre a ideia de negócio e da interface com o utilizador, no desenvolvimento ou alteração de módulos do sistema caso seja necessário; esclarecem processos e procedimentos que respeitem o produto a desenvolver, seja módulo ou nova funcionalidade, validam a respetiva especificação funcional e protótipos, e produto final;
- Comissão de Utilizadores: é constituída por Gestores de Informação (GI),que são pessoas próximas dos órgãos de gestão, de modo a terem um conhecimento profundo da estrutura orgânica, missão e objetivos das entidades. O GI deverá assegurar [55]:
  - A organização dos conteúdos no SIGARRA da sua UO;
  - A identificação dos Produtores de Informação locais;

  - O apoio aos Produtores de Informação locais;
  - A difusão de informação interna sobre o SIGARRA;
  - As boas práticas na utilização do sistema;
  - A colaboração para a validação de especificações de alterações ou de novos desenvolvimentos a realizar no SIGARRA;
  - A colaboração na produção de guias, manuais e outros documentos de apoio para o SIGARRA, na perspetiva dos utilizadores.

- Desenvolvimento e Inovação: este departamento é da responsabiblidade da Faculdade de Engenharia, e são responsáveis por [55]:

  - Elaboração de especificações.
  - Análise e Desenho:
    * Definição de requisitos e elaboração de documentação de âmbito funcional;
    * Conceção de modelos de dados e elaboração de documentação técnica;
    * Especificação da interoperabilidade com outros sistemas, quando aplicável;
    * Desenho de interfaces.
  - Desenvolvimento de código (localização em duas línguas):
    * Desenvolvimentos de ferramentas de produtividade;
    * Prototipagem;
    * Implementação dos modelos de dados, API e interfaces.
  - Manutenção:
    * Correção de falhas (bugs) ou inconformidades do código;
    * Novas versões de módulos ou funcionalidades.
  - Controlo de qualidade de software:
    * Definição de plano de testes;
    * Realização de testes;
    * Conformidade com normas de acessibilidade e usabilidade;
    * Produção de relatório de controlo de qualidade.
  - Gestão de documentação:
    * Ajuda on-line;
    * Documentação de *releases*.
  - Formação de formadores:
    * Formação de equipas técnicas de consultadoria e/ou suporte.

- Infraestrutura Técnica: a gestão deste departamento é da responsabilidade do Centro de Recursos e Serviços Comuns da Universidade do Porto (CRSCUP); devem gerir e administrar toda a infraestrutura técnica e física e lógica de suporte aos vários ambientes necessários ao projeto SIGARRA, assegurando elevados níveis de desempenho, disponibilidade e fiabilidade destes ambientes.

- Consultadoria e Suporte: em conjunto com os GI e equipas de suporte locais, de casa UO, no sentido de assegurar o conhecimento dos módulos do sistema e a sua utilização, de acordo com os requisitos estabelecidos e norma de boas práticas; com a colaboração da Unidade de Desenvolvimento e Inovação asseguram a correção de erros e inconformidades nos módulos e funcionalidades do sistema. São responsáveis por [55]:

  - Identificar erros e inconformidades do código, de acordo com os processos e procedimentos organizacionais;
  - Pronunciar-se sobre a aceitação de desenvolvimentos realizados;
  - Gestão de perfis e o controlo de acessos ao sistema;
  - Controlo de qualidade de dados;
  - Consultadoria aos Gestores de Informação para a boa utilização do sistema;
  - Suporte (helpdesk) de 2.ª linha e a formação às equipas locais, incluindo a produção de FAQ;
  - Promoção de boas práticas e o estímulo à utilização dos vários módulos do sistema;
  - Produção de materiais de apoio à utilização e divulgação do sistema;
  - Produção e atualização de conteúdos sobre o sistema, no portal TIC da U.Porto;
  - Produção de estatísticas e de relatórios de apoio à gestão do projeto.

Embora o Projeto SIGARRA seja organizado e bem estruturado, possibilitando o envolvimento de todas as entidades da UP, não possui qualquer tipo de certificação em normas da qualidade, é uma organização pouco madura e com algumas lacunas a nível de documentação e de organização. A equipa da Unidade de Desenvolvimento e Inovação é composta por 23 pessoas, não existe organização interna, isto é, não são definidos papéis associados a cada elemento da equipa.

O PRODUTO

O processo de criação do SIGARRA foi longo, e durante esse percurso destacam-se a aplicação de Gestão Académica criada pela Reitoria (1992), o SiFEUP, um sistema de informação académico desenvolvido pela Faculdade de Engenharia da Universidade do Porto (1996), e a aplicação de Gestão de Recursos Humanos para o serviço de recursos humanos (1999). Em 2003, foram fundidos estes três componentes no âmbito de um projeto levado a cabo pela FEUP e pela Universidade Digital, com o objetivo de disponibilizar o SiFEUP a todas as entidades da UP.

Organismo	Data Início
Faculdade de Arquitetura	Setembro de 2004
Faculdade de Belas Artes	Julho de 2004
Faculdade de Ciências	Setembro de 2007
Faculdade de Ciências da Nutrição e Alimentação	Outubro de 2003
Faculdade de Desporto	Julho de 2004
Faculdade de Direito	Janeiro de 2004
Faculdade de Economia	Outubro de 2003
Faculdade de Engenharia	Outubro de 1996
Faculdade de Farmácia	Março de 2004
Faculdade de Letras	Setembro de 2003
Faculdade de Medicina	Janeiro de 2005
Faculdade de Medicina Dentária	Outubro de 2003
Faculdade de Psicologia e de Ciências da Educação	Setembro de 2003
Instituto de Ciências Biomédicas Abel Salazar	Janeiro de 2004
Centro de Desporto	Abril de 2013
Centro de Recursos e Serviços Comuns	Março de 2013
Instituto de Recursos e Iniciativas Comuns Ícone para dar informação	Março de 2003
Reitoria Ícone para dar informação	Janeiro de 2005
Reitoria / Instituto de Recursos e Iniciativas Comuns Ícone para dar informação	Outubro de 2006
Serviços de Acção Social	Março de 2006
Universidade do Porto	Setembro de 2005

Figura 4.3: Instâncias do SIGARRA na U.PORTO

O SIGARRA é composto por duas componentes de *backoffice* - a Gestão Académica (GA) e a Gestão de Recursos Humanos (GRH) - e uma componente de *frontoffice* - Sistema de Informação (SI) [56]. Cada uma destas componentes é constituída por módulos, que em conjunto compõem o SIGARRA, estes módulos são organizados por áreas temáticas.

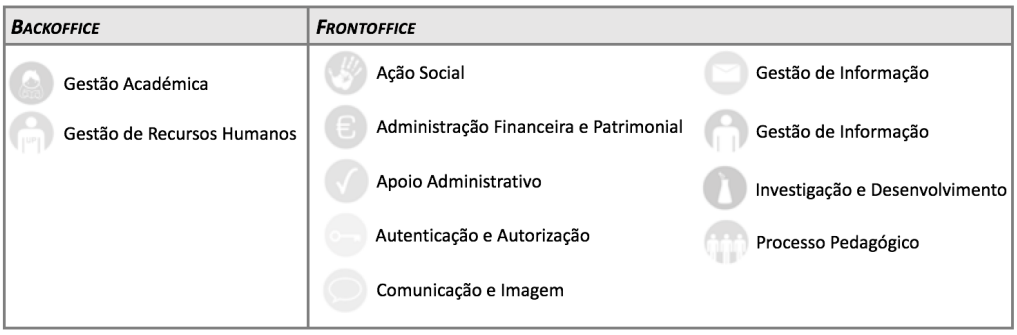

Figura 4.4: Áreas temáticas do SIGARRA na U.PORTO

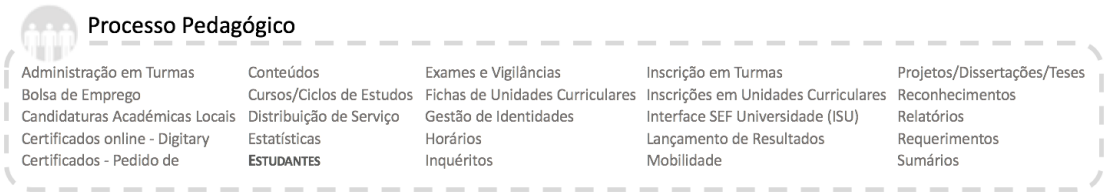

Figura 4.5: Módulos da área temática Processo Pedagógico

A área temática Processo Pedagógico é constituída por 25 módulos (figura 4.5); os defeitos que vão ser analisados durante a aplicação prática pertencem ao módulo Estudantes.

4.2 DCA no SIGARRA

4.2.1 Selecionar uma amostra

A amostra compreendeu os defeitos do módulo Estudantes; este é usado para a visualização de dados académicos e pessoais dos estudantes que ainda frequentam a universidade ou que já frequentaram. A informação é obtida a partir dos dados registados na Gestão Académica e encontra-se organizada por curso/ciclo de estudos que o estudante frequente, ou tenha frequentado.

Como mostra a figura 4.6, inclui ainda informação referente a: [57]

- ao percurso/académico;
- à posição no plano;
- às unidades curriculares com inscrição;
- aos estatutos e regimes de frequência;
- aos reconhecimentos;
- aos requerimentos;
- ao horário;
- aos certificados.

Figura 4.6: Ficha de estudante

Foram cedidos pela equipa do Projeto SIGARRA 20 registos de defeitos, reportados pelos utilizadores: estudantes, pessoal administrativo e professores. Os defeitos são registados na plataforma JIRA, que tem como objetivo *tracking* de defeitos e gestão de projetos. Cada registo tem os seguintes campos:

- *Project*: o módulo a que pertence o erro;
- *Priority*: classificação baseada na importância e a urgência de o resolver (*Urgent, High, Medium, Low* e *Minor*);
- *Severity*: impacto que o defeito tem no projeto (*Blocker, Critical, Major, Normal* e *Trivial*) ;
- *Tester*: encarregado por efetuar os testes se o erro persiste mesmo depois de corrigido o defeito;
- *Developer*: responsável pelo desenvolvimento do código;
- *Description*: descrição do erro encontrado e os passos para o reproduzir.

4.2.2 Classificar os defeitos selecionados

O passo seguinte consiste em classificar ou agrupar os defeitos. Ajuda a identificar "*clusters*" em que os erros sistemáticos são suscetíveis de serem encontrados. Existem três aspetos úteis para a classificação:

- Quando é que o defeito foi inserido?

- Quando é que foi detetado?

- Que tipo de erro foi cometido ou defeito introduzido?

Os primeiros dois pontos correspondem a atividades ou fases do processo de desenvolvimento de software. O último, incide na natureza do trabalho realizado e a oportunidade de cometer erros [28]. Na análise causal é usado ODC; é um dos métodos para analisar e classificar os defeitos.

Este tipo de método, preenche a lacuna entre o modelo estatístico e a análise causal, classificando cada defeito com base na semântica da sua correção [58].

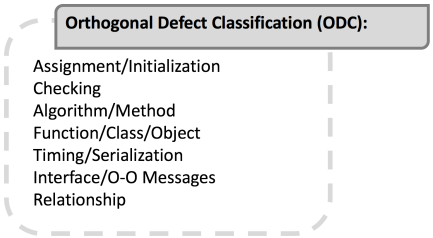

Figura 4.7: Classificação ODC

Cada defeito foi estudado de modo a identificar o atributo a que pertenciam dentro do conceito ODC. Depois de serem classificados foram agrupados consoante esse mesmo atributo. Durante a análise foi encontrado um defeito que acabou por ser classificado como *not bug*.

Defect Type	Amount	Cumulative	Cumulative %
Assignment/Initialization	11	11	58%
Relationship	6	17	89%
Checking	2	19	100%
Algorithm/Method	0	19	100%
Function/Class/Object	0	19	100%
Timing/Serialization	0	19	100%
Interface/O-O Messages	0	19	100%
Not a bug	1		
Total	**19**		

Figura 4.8: Classficação dos defeitos

Gráficos de Pareto permitem ordenar a frequência das ocorrências, da maior para a menor, permitindo priorizar os defeitos pois podem existir muitos problemas sem importância perante outros mais graves. Para uma melhor visualização e identificação das causas mais importantes que originaram os defeitos no módulo Estudantes, foi criado um gráfico de Pareto (4.9).

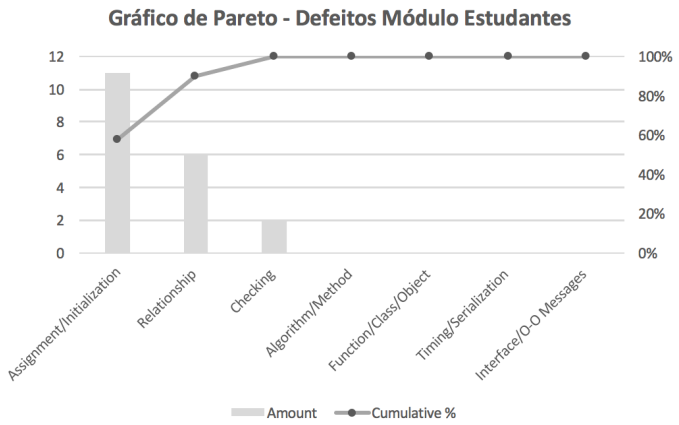

Figura 4.9: Gráfico de Pareto

Assim sendo e analisando a figura 4.9, os defeitos do módulo em estudo recaem apenas em três das sete categorias do ODC, *Assignment/Initialization* com aproximadamente 58%, *Checking* com 32% perfazendo uma combinação de 89% e por fim *Relationship* com 11%.

4.2.3 Identificar erros sistemáticos

Os defeitos foram analisados individualmente com o apoio de um elemento da equipa de desenvolvimento do SIGARRA, de modo a serem detetados os erros que os provocaram. No seguimento desta ação, foram identificados os erros sistemáticos para cada conjunto de defeitos. Na figura 4.10, pode-se observar que foram encontrados os seguintes erros sistemáticos:

- *User Interface (UI), settings GESSI* da categoria *Assignment/Initialization*;
- *User Interface, Application Programming Interface (API)* da categoria *Assignment/Initialization*;
- API, *access to data* da categoria *Relationship*;
- API, *data rules* da categoria *Relationship*;
- API, *validation rules, local settings* da categoria *Checking*;

#	ID	Defect Category	Error	Systematic Errors
1	SIFEST-1264	Assignment/Initialization	User Interface, bad pratice	
2	SIFEST-722	Assignment/Initialization	User Interface, settings GESSI	User Interface, settings GESSI
3	SIFEST-685	Assignment/Initialization	User Interface, settings GESSI	
4	SIFEST-684	Assignment/Initialization	Settings GESSI	
5	SIFEST-2288	Assignment/Initialization	User Interface, Application Programming Interface, validation	
6	SIFEST-2303	Assignment/Initialization	User Interface, CSS	
7	SIFEST-1313	Assignment/Initialization	User Interface, Application Programming Interface	User Interface, Application Programming Interface
8	SIFEST-752	Assignment/Initialization	User Interface, Application Programming Interface	
9	SIFEST-1149	Assignment/Initialization	User Interface, settings GESSI, bad pratice	
10	SIFEST-724	Assignment/Initialization	Application Programming Interface, settings GESSI	
11	SIFEST-1142	Assignment/Initialization	Application Programming Interface	
12	SIFEST-2365	Relationship	Application Programming Interface, access a data (data rules)	Application Programming Interface, access to data
13	SIFEST-2351	Relationship	Application Programming Interface, access to data (validation)	
14	SIFEST-919	Relationship	Application Programming Interface, data rules (write control DB)	Application Programming Interface, data rules
15	SIFEST-2241	Relationship	Application Programming Interface, data rules	
16	SIFEST-2209	Relationship	Application Programming Interface, data rules	
17	SIFEST-699	Relationship	Application Programming Interface, data rules	
18	SIFEST-1266	Checking	Application Programming Interface, validation rules, local settings	Application Programming Interface, validation rules, local settings
19	SIFEST-1308	Checking	Application Programming Interface, validation rules, local settings	

Figura 4.10: Tabela de erros sistemáticos do SIGARRA

4.2.4 Identificar as principais causas

Depois de encontrados os erros sistemáticos, o próximo passo foi identificar as principais causas para que a sua ocorrência. Assim sendo, para os defeitos categorizados como *Assignment/Initialization* os erros sistemáticos foram *User Interface, settings GESSI* e *Application Programming Interface*. Ao analisar a figura 4.11, pode-se observar a categoria que pertencem e as suas causas.

Defect Category	Error	Systematic Errors	Category	Cause
Assignment/Initialization	User Interface, settings GESSI	User Interface, settings GESSI	Method	Absence of user interface testing
Assignment/Initialization	User Interface, settings GESSI			
Assignment/Initialization	User Interface, Application Programming Interface	User Interface, Application Programming Interface	Method	Absence of automated testing
Assignment/Initialization	User Interface, Application Programming Interface			

Figura 4.11: Categoria *Method* e causas principais

Por sua vez, na figura 4.12 estão descritas as causas principais para os defeitos da categoria *Relationship*, bem como os seus erros sistemáticos (*Application Programming Interface, access to data* e *data rules*). Estes últimos encontram-se divididos em duas categorias *Method* e *Tools*.

Defect Category	Error	Systematic Errors	Category	Cause
Relationship	Application Programming Interface, access a data (data rules)	Application Programming Interface, access to data	Method	Absence of automated testing
Relationship	Application Programming Interface, access to data (validation)			
Relationship	Application Programming Interface, data rules (write control DB)	Application Programming Interface, data rules	Tools	Lack of documentation of APIs
Relationship	Application Programming Interface, data rules			
Relationship	Application Programming Interface, data rules			
Relationship	Application Programming Interface, data rules			

Figura 4.12: Categoria *Method* e *Tools* e causas principais

Por último, encontram-se os defeitos da categoria *Checking* e as causas que os provocam. Na figura 4.13 pode-se observar que o erro sistemático para estes dois defeitos é *Application Programming Interface, validation rules, local settings* e que foram categorizado como *Tools*.

Defect Category	Error	Systematic Errors	Category	Cause
Checking	Application Programming Interface, validation rules, local settings	Application Programming Interface, validation rules, local settings	Tools	Lack of documentation on the business rule
Checking	Application Programming Interface, validation rules, local settings			

Figura 4.13: Categoria *Tools* e causas principais

4.2.5 Desenvolver propostas de ação

Embora a análise causal neste passo se concentre em encontrar soluções para as causas dos erros sistemáticos, neste caso prático, e em conjunto com o responsável pela parceria, foi decidido como uma mais valia para a equipa do projeto encontrar propostas para todos os defeitos da amostra. Assim sendo, na figura 4.14 estão descritas ações a pôr em prática para auxiliar a equipa do Projeto SIGARRA a prevenir os defeitos.

#	ID	Defect Category	Error	Action
1	SIFEST-1264	Assignment/Initialization	User Interface, bad pratice	Conduct UI tests
2	SIFEST-722	Assignment/Initialization	User Interface, settings GESSI	Conduct UI tests
3	SIFEST-685	Assignment/Initialization	User Interface, settings GESSI	Conduct UI tests
4	SIFEST-684	Assignment/Initialization	Settings GESSI	Conduct module utilization tests with different profiles (Including Administrators/Support teams)
5	SIFEST-2288	Assignment/Initialization	User Interface, Application Programming Interface, validation	Conduct utilization and automatic tests
6	SIFEST-2303	Assignment/Initialization	User Interface, CSS	Conduct UI tests
7	SIFEST-1313	Assignment/Initialization	User Interface, Application Programming Interface	Conduct UI tests
8	SIFEST-752	Assignment/Initialization	User Interface, Application Programming Interface	Creation of external documentation (business rules)
9	SIFEST-1149	Assignment/Initialization	User Interface, settings GESSI, bad pratice	Conduct UI tests
10	SIFEST-724	Assignment/Initialization	Application Programming Interface, settings GESSI	Creation of external documentation (business rules)
11	SIFEST-1142	Assignment/Initialization	Application Programming Interface	Creation of external documentation (business rules)
12	SIFEST-2365	Relationship	Application Programming Interface, access a data (data rules)	Automatic tests utilization; Creation of extra abstraction layers in order to cluster identical actions with different origins within the system
13	SIFEST-2351	Relationship	Application Programming Interface, access to data (validation)	Automatic tests utilization; Creation of extra abstraction layers in order to cluster identical actions with different origins within the system
14	SIFEST-919	Relationship	Application Programming Interface, data rules (write control DB)	Creaton of APIs internal documentation (and business rules)
15	SIFEST-2241	Relationship	Application Programming Interface, data rules	Creaton of APIs internal documentation (and business rules)
16	SIFEST-2209	Relationship	Application Programming Interface, data rules	Creation of external documentation (business rules)
17	SIFEST-699	Relationship	Application Programming Interface, data rules	Automatic tests utilization; Creation of APIs internal and external documentation (and business rules)
18	SIFEST-1266	Checking	Application Programming Interface, validation rules, local settings	Creation of external documentation (business rules)
19	SIFEST-1308	Checking	Application Programming Interface, validation rules, local settings	Creation of external documentation (business rules)

Figura 4.14: Propostas de ação

Na figura 4.14 pode-se verificar que, muitas das ações são iguais ou similares mesmo pertencendo a defeitos de diversas categorias e de diferentes erros. É de notar uma grande carência no que toca aos testes do produto e na documentação interna ou externa do projeto. Assim sendo e como indica a figura 4.15, foram encontradas ações de melhoria que vão ajudar a equipa de desenvolvimento do SIGARRA a prevenir e a detetar defeitos antes que estes cheguem ao cliente.

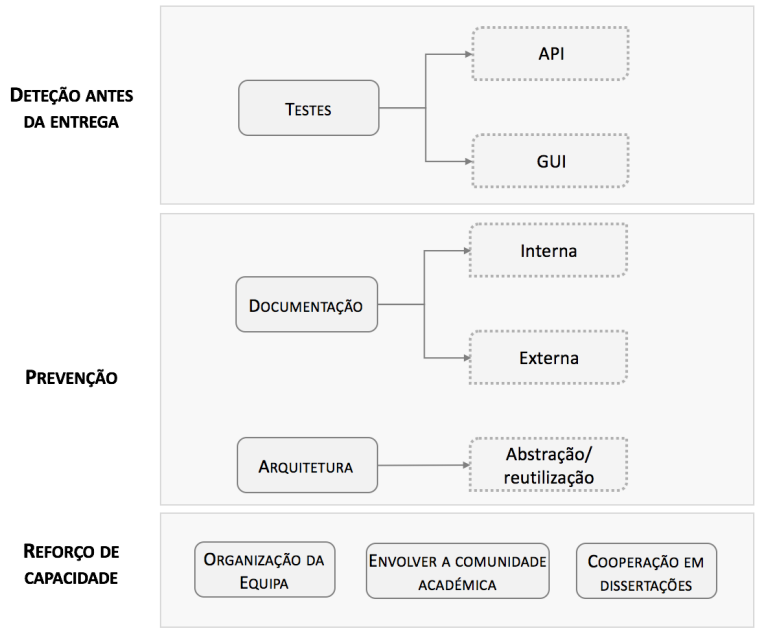

Figura 4.15: Ações de melhoria

TESTES AUTOMÁTICOS

Todas as empresas de desenvolvimento de software testam os seus produtos, contudo existem sempre defeitos que passam despercebidos. As empresas esforçam-se para os encontrar antes do produto ser entregue ao cliente, mas muitas das vezes eles reaparecem, mesmo com os melhores processos de testes manuais. O uso de testes automáticos é a melhor maneira de aumentar a eficácia, eficiência e cobertura de testes; são essenciais em qualquer área de testes e qualidade de software dentro de uma empresa. São vários os benefícios destes tipo de teste [59]:

- Aumento da produtividade das pessoas que testam o software;
- Durações reduzidas nas fases de teste;
- Redução do custo de manutenção de software;
- Aumento da eficácia de casos de teste.

• *Application Programming Interface*

Na programação, uma API é um conjunto de rotinas, protocolos e ferramentas para a construção de software, que podem ser executadas por outras aplicações. O código desenvolvido pelos *developers* cria uma ligação entre as APIs existentes para tirar partido da sua funcionalidade [60]. Os teste de API são um tipo de teste de software que envolve interface gráfica do aplicativo de teste, fazem parte de testes de integração para determinar se as APIs correspondem as expectativas de funcionalidade, desempenho, segurança e credibilidade.

- Responsabilidade: este testes têm de ser conduzidos por um elemento da equipa de desenvolvimento, que deve ser responsável por criar [47]:

Estratégia de teste: que tipos de testes e o número de testes que devem de ser realizados para encontrar os defeitos que possam estar no software;

Plano de teste: as tarefas que vão ser necessárias para pôr em prática a estratégia de teste;

Casos de teste: casos que foram preparados com antecedência, sob a forma de exemplos detalhados para verificar se o software responde às necessidades;

Dados do teste: dados que são introduzidos durante a execução dos testes;

Ambiente de teste: ambiente em que vão ser conduzidos os testes.

• *Graphical User Interface*

A interface gráfica de uma aplicação nos dias de hoje é o meio de interação entre utilizador final e o produto de software. A qualidade da GUI é um fator determinante na escolha de usar um sistema ou não, logo deve de ser graficamente atraente, ser eficiente e eficaz, *user-friendly* e livre de defeitos [61]. Nesse sentido, os testes de GUIs têm como objetivo encontrar defeitos nas interfaces gráficas. No entanto, existem ainda poucas ferramentas e técnicas disponíveis no auxílio deste tipo de teste [62].

Assim sendo, os testes de GUIs servem para salvaguardar que foram cumpridas todas as especificações e para detetar se a aplicação está a funcionar de acordo com os requisitos. Envolvem a execução de algumas tarefas e a comparação do seu resultado com o resultado esperado, podendo ser realizados manualmente ou de forma automática [63].

- Responsabilidade: a mesma que os testes de API e com os mesmo procedimentos, no entanto uma sugestão a aplicar no projeto SIGARRA seria a técnica *capture and replay*, que consiste em capturar e gravar a interação do utilizador (como cliques e movimentos do rato, entradas do teclado, ...) ou do *tester* com a aplicação e reproduzi-la [64]. Essa interação funciona como uma entrada, em que são registadas as ações feitas pelo *tester* (script) e depois reproduzidas de forma automática. O resultado gerado é comparado com o resultado final e se forem iguais então a GUI está livre do erros a que está a ser testado, senão são analisadas as diferenças entres os resultados e investigam-se as causas [65].

- Ferramentas *capture and replay*: Selenium, Sahi, Watir, Badboy, Jubula, QF-Test, Squish, Test Studio, eggPlant, Squish, TestComplete e Testing Anywhere.

DOCUMENTAÇÃO

Nos grande projetos de software, a documentação começa a ser criada antes do início do desenvolvimento. Elaboram-se documentos que auxiliam na gestão do desenvolvimento e na manutenção do projeto, tais como, planificação, calendários, processos de qualidade, processos de organização e standards do projeto. São também produzidos documentos que apoiam o processo de desenvolvimento do produto; este tipo de documentos descreve o produto que está a ser desenvolvido, tais como regras de negócio e os seus requisitos, definem as características exigidas do sistema esperado [66].

• Interna

APIs bem documentadas melhoram a experiência para os *developers*, e são um requisito essencial para o sucesso de uma API. Este tipo de documentação, é uma escrita técnica com instruções e descrições, recorrendo a exemplo de como usar de forma eficiente a API que está a ser explorada [67].

- Responsabilidade: este relatório técnico deve de ser escrito por uma pessoa da equipa de desenvolvimento do SIGARRA, que deve criar instruções, procedimentos e exemplos para a API.

• Externa

Os requisitos são descrições dos serviços que um sistema de software deve fornecer, e as restrições sob as quais ele deve operar. Estes podem variar desde afirmações abstratas de alto nível dos serviços ou até restrições do sistema para especificações funcionais. Este tipo de documentação descreve o processo de estabelecer quais os serviços que o cliente requer e as restrições sob as quais o sistema deve de ser desenvolvido e operado.

- Responsabilidade: deve de ser escrito por uma pessoa da equipa de desenvolvimento do SIGARRA, deve de especificar os requisitos do usuário ou utilizador, requisitos do sistema e design da aplicação.

ARQUITETURA

A pressão dos mercados para reduzir o tempo de desenvolvimento dos produtos exige das equipas de desenvolvimento a habilidade de construir sistemas com qualidade, e para que isso aconteça existe um esforço para reutilizar artefactos que foram criados anteriormente. Com o reaproveitamento em níveis mais elevados de abstração, os benefícios da reutilização podem ser atingidos desde o início do ciclo de vida do software [68].

ORGANIZAÇÃO DA EQUIPA

Visto que dentro da equipa de desenvolvimento do SIGARRA não existem papeis atribuídos e existe falta de mão de obra, é necessário criar uma estrutura de funcionamento dentro da equipa, definindo *developers* e *testers*. Uma das soluções seria o contrato a tempo parcial de estudantes da Faculdade de Engenharia, que necessitem de ajuda a nível financeiro, mas que tenham interesse na área de testes e que queiram enriquecer o curriculum.

ENVOLVER A COMUNIDADE ACADÉMICA

Envolvimento de utilizadores do SIGARRA na fase de testes, reunir um pequeno grupo de pessoas (estudantes, serviços académicos) que utilizem as variadas opções de perfis disponíveis do SIGARRA para efetuar os testes nos módulos a que tenham acesso.

COOPERAÇÃO COM DISSERTAÇÕES

Existem lacunas no projeto SIGARRA, como por exemplo falta de documentação ou falta de um plano de testes automáticos; uma das formas de atender a esta necessidade seria a disponibilização de dissertações que pudessem ser úteis para o SIGARRA e de interesse académico para os

alunos.

4.2.6 Documentar os resultados da reunião

É necessário um registo da reunião para assegurar que todas as ações de melhoria são implementadas. No documento as ações são priorizadas, planeadas e programadas ações similares, controlo do progresso da implementação e da eficácia das ações. No caso desta aplicação prática foram enviados ao responsável desta parceria o conjunto de ações de melhoria e sugestões que foram desenvolvidas para o SIGARRA.

Capítulo 5

Conclusões e Trabalho Futuro

A análise causal de defeitos implica, de acordo com as características do projeto, um conhecimento da relação entre a causa e o efeito de variáveis ou eventos. É necessário perceber quais as causas que provocam certos tipos de defeitos, para os poder prevenir. A análise causal tem-se mostrado capaz de melhorar processos e reduzir a introdução de defeitos em diferentes contextos organizacionais. No entanto, e de acordo com a pesquisa efetuada durante a preparação da dissertação, as abordagens encontradas para aplicar a análise causal de defeitos (ACD) descrevem as várias atividades a serem realizadas, mas fornecem pouco detalhe sobre as tarefas que devem ser executadas no contexto dessas atividades e sobre as técnicas que podem apoiar a implementação eficiente destas tarefas.

A Critical Manufacturing e o projeto SIGARRA possuem ferramentas de registo de erros, TFS e JIRA respetivamente, no entanto nenhuma das empresas dá importância ao que provocou o erro mas sim a qual o erro detetado. O pensamento geral é "foi reportado um erro, é necessário corrigi-lo e registar na plataforma que está resolvido"; não existe uma preocupação por parte das empresas em saber o que o provocou, mas apenas que está solucionado. O que, consequentemente, fez com que as amostras de defeitos cedidas pelas empresas tivessem de ser retornadas para que fossem registadas quais as correções aplicadas a cada erro, provocando alguns atrasos na realização deste estudo e exigido um esforço extra das organizações envolvidas.

Posteriormente, foram efetuadas as respetivas análises e encontradas as principais causas para a ocorrência dos defeitos selecionados. Neste tópico, constatou-se um grande interesse por parte das empresas, pois ambas concordaram que muitos dos defeitos encontrados podiam ser prevenidos no futuro. No entanto, o projeto SIGARRA, sendo uma estrutura com mais carência a nível de metodologias e processos organizacionais, foi a que demonstrou mais recetividade e interesse no estudo. Concluiu-se que as empresas, independentemente do nível de maturidade em que se encontram, têm interesse em apostar e investir na melhoria da qualidade dos produtos que desenvolvem.

No seguimento da análise resultou um conjunto de soluções ou propostas de melhoria a implementar em cada um dos casos estudados. É de notar que no caso da CM, as ações de melhoria pertencem ao foro técnico. No SIGARRA, as sugestões recaem mais na fase de testes e de docu-

mentação, o que vai de encontro com o que foi dito anteriormente, em empresas que se encontram num nível de maturidade mais baixo existe uma carência de processos que descrevam procedimentos, métodos e ferramentas a utilizar.

Para trabalho futuro, e depois de aplicadas as ações de melhoria encontradas para cada caso, seria útil recolher o *feedback* das empresas envolvidas e efetuar um novo levantamento de defeitos para analisar. Uma possível solução para contornar a adversidade inicial da análise, poderia passar por ser pedido às empresas que, ao alterar o estado do defeito, introduzissem também qual o erro encontrado e o que foi corrigido. Uma outra sugestão poderia ser a classificação do erro consoante a classificação escolhida pela empresa por parte do elemento da equipa que o corrigisse, visto ser ele a pessoa mais inteirada do mesmo, isto é, o passo dois na ACD passava a ser da responsabilidade de quem corrigisse o erro.

Conclui-se, então, que a análise causal de defeitos embora seja pouco utilizada e explorada é uma mais valia para qualquer empresa. Adicionalmente, concluiu-se também ser necessário uma preparação prévia das empresas para aplicar este tipo de abordagem, o que mostrou ser bastante benéfica.

Referências

[1] Critical e Manufacturing. Introdução - A Empresa. URL: http://www.criticalmanufacturing.com/pt/company/overview.

[2] Universidade do Porto. SIGARRA - Universidade do Porto, 2015. URL: https://sigarra.up.pt/up/pt/web_page.inicial.

[3] Vincent K. Omachonu e Joel E. Ross. Management of Process Quality. Em *Principles of Total Quality 3rd edition*, chapter 6. 2005.

[4] James Mchale, Tim Chick, e Eugene Miluk. Implementation Guidance for the Accelerated Improvement Method. (December):108, 2010. URL: http://www.sei.cmu.edu/reports/10sr032.pdf.

[5] José Ricardo Rigoni. Qualidade do produto X Qualidade do processo - Garantia da Qualidade e Controle da Qualidade. URL: http://www.totalqualidade.com.br/2012/09/qualidade-do-produto-x-qualidade-do.html.

[6] Vincent K. Omachonu e Joel E. Ross. Total Quality Management and the Revival of Quality in the U.S. Em *Principles of Total Quality 3rd edition*, chapter 1. 2005.

[7] Philip B. Crosby. Cost of quality. Em *Quality is Free*, chapter 7. 1979.

[8] Vincent K. Omachonu e Joel E. Ross. The cost of quality. Em *Principles of Total Quality 3rd edition*, chapter 11. 2005.

[9] David A. Garvin. The concept of Quality. Em *Managing Quality- The Strategic and Competitive Edge*, chapter Part 1. 1988.

[10] Jeff Tian. What Is Software Quality? Em *Software Quality Engineering*, chapter 2, páginas 15–26. 1990.

[11] Potential Contamination, O F Jet, e Fuel With. Product Quality. (20):1–4, 2008.

[12] Daniela C C Peixoto, Vitor A Batista, Gustavo M Rocha, e Clarindo Isaías P S. Uma Experiência de Melhoria de Processo utilizando a Análise Causal de Defeitos. 2008.

[13] David Hoyle. Role, origins and application of ISO 9000. Em *ISO 9000: Quality Systems Handbook, 4th Edition*, chapter 3, páginas 80–114. Butterworth-Heinemann, 2001.

[14] ISO. Quality management principles. 2012. URL: http://www.iso.org/iso/qmp_2012.pdf.

[15] ISO. Iso 9001 - How to use it. 2015.

[16] Vincent K. Omachonu e Joel E. Ross. Introduction to Six Sigma. Em *Principles of Total Quality 3rd edition*, chapter 27. 2005.

[17] ISixSigma. DMAIC Versus DMADV. URL: http://www.isixsigma.com/new-to-six-sigma/design-for-six-sigma-dfss/dmaic-versus-dmadv/.

[18] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, e Charles V Weber. Capability Maturity Model for Software. 1993. URL: https://resources.sei.cmu.edu/asset_files/TechnicalReport/1993_005_001_16211.pdf.

[19] Paula Donegan, Liane Bandeira, Márcia Sampaio, Carlo Giovano Pires, e Arnaldo Dias Belchior. Métricas de Software: Um Mapeamento entre Six Sigma e CMMI. *VII Simpósio Internacional de Melhoria de Processo de Software*, página 24, 2005. URL: http://www.simpros.com.br/simpros2005/upload/A10_3_artigo14725.pdf.

[20] Kshirasagar Naik e Priyadarshi Tripathy. Maturity Models. Em *Software Testing and Quality Assurance, Theory and Practice*, chapter 18, páginas 546–579. 2008.

[21] Mark C Paulk. A History of the Capability Maturity Model ® for Software. *Total Quality Management*, 2001.

[22] CMMI Product Team. CMMI for Development, Version 1.3. Relatório técnico, 2010.

[23] CMMI Architecture Team. Introduction to the Architecture of the CMMI Framework. Relatório técnico, 2007.

[24] CMMI Product Team. CMMI for Acquisition, Version 1.3. Relatório técnico, Carnegie Mellon University, 2010.

[25] CMMI Product Team. CMMI for Services, Version 1.3. Relatório técnico, 2010.

[26] Mary Beth Chrissis, Mike Konrad, e Sandy Shrum. Process Area Components. Em *CMMI for Development - Guidelines for Process Integration and Product Improvement, 3rd Edition*, chapter 2. 2011.

[27] Cmmi Services. CMMI® for Services, Version 1.3 CMMI-SVC, V1.3. (November), 2010.

[28] David N. Card. Defect Analysis: Basic Techniques for Management and Learning. *Advances in Computers*, 65, 2005.

[29] Marcos Kalinowski e Guilherme Horta Travassos. Uma Abordagem Probabilística para Análise Causal de Defeitos de Software. *XI Simposio Brasileiro de Qualidade de Software*, 2011.

[30] Padmanabhan Santhanam. Orthogonal Defect Classification. URL: http://researcher.watson.ibm.com/researcher/view_group.php?id=480.

[31] IBM. Orthogonal Defect Classification v5.2 - for Software Design and Code. 2013.

[32] Norm Bridge e Corinne Miller. Orthogonal Defect Classification Using Defect Data to Improve Software Development Norm Bridge Motorola Corporate Software Center Motorola GSM Products Division Arlington Heights , Illinois. 1998.

[33] Eileen C. Forrester, Brandon L. Buteau, e Sandy Shrum. *CMMi for Services - Guidelines for Superior Service, version 1.3*. Addison-Wesley Professional, 2011.

[34] Marilyn K. Hart e Robert F. Hart. *Statistical Process Control Techniques*. Statit Software, 2007.

[35] InfinityQS. What is Statistical Process Control (SPC)? URL: http://www.infinityqs.com/resources/what-is-spc.

[36] Vincent K. Omachonu e Joel E. Ross. The Seven Basic Quality Control Tools. Em *Principles of Total Quality 3rd edition*, chapter 14. 2005.

[37] Marcos Kalinowski. Melhorando Processos de Software através de Análise Causal de Defeitos. *Engenhara de Software Magazine*, 2015.

[38] Marcos Kalinowski, Guilherme H Travassos, e David N Card. Guidance for Efficiently Implementing Defect Causal Analysis. *Proceedings: Brazilian Software Quality Symposium, June 2008*, 2008.

[39] Jeff Tian. Defect Classification and Analysis. Em *Software Quality Engineering*, chapter 20, páginas 339–351. 1990.

[40] Marcos Kalinowski, Emilia Mendes, e Guilherme Horta Travassos. Automating and Evaluating Probabilistic Cause-Effect Diagrams to Improve Defect Causal Analysis. 2011.

[41] Critical Manufacturing. Critical Manufacturing Quality Manual. Relatório técnico.

[42] ISTQB Exam Certification. What is the cost of defects in software testing? URL: http://istqbexamcertification.com/what-is-the-cost-of-defects-in-software-testing/.

[43] Critical Manufacturing. CMNavigo MES - The Future of Manufacturing Execution Systems, 2014. URL: http://www.criticalmanufacturing.com/en/cmnavigo-mes/overview.

[44] Critical Manufacturing. cmNavigo - A mighty Manufacturing Execution System. URL: http://www.crossborder-technologies.com/node/71.

[45] Microsoft. Build and test integration field reference, 2016. URL: https://msdn.microsoft.com/en-us/library/dd997786.aspx.

[46] Microsoft. Visual Studio, 2016. URL: https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx.

[47] David Lile Brown. The Five Essentials for Software Testing, 2014. URL: http://www.isixsigma.com/industries/software-it/five-essentials-software-testing/.

[48] Wikipédia. Wikipédia - Test Driven Development. URL: https://pt.wikipedia.org/wiki/Test_Driven_Development.

[49] Roy W. Miller e Christopher T. Collins. Acceptance testing. 2001.

[50] Tom Huston. What is code review? URL: http://smartbear.com/learn/code-review/what-is-code-review/.

[51] Alistair Cockburn e Laurie Williams. The Costs and Benefits of Pair Programming. 2000.

[52] Universidade do Porto. Wikipédia - Universidade do Porto, 2015. URL: https://pt.wikipedia.org/wiki/Universidade_do_Porto.

[53] Universidade do Porto. A Universidade do Porto em Números 2013, 2013. URL: https://sigarra.up.pt/up/pt/web_base.gera_pagina?p_pagina=recursoseservi%C3%A7osdau.porto-recursosinformacionais:factosen%C3%BAmeros.

[54] Universidade do Porto. SIGARRA - Sistema de Informação para Gestão Agregada dos Recursos e dos Registos Académicos. URL: https://sigarra.up.pt/up/pt/web_base.gera_pagina?P_pagina=2418.

[55] Universidade do Porto. Projeto SIGARRA da U.Porto - Responsabilidades e Competências. URL: https://sigarra.up.pt/up/pt/web_base.gera_pagina?P_pagina=1004288#responsab.

[56] Universidade do Porto. SIGARRA - Módulos. URL: https://sigarra.up.pt/up/pt/web_base.gera_pagina?P_pagina=1000325.

[57] Universidade do Porto. SIGARRA - Estudantes. URL: https://sigarra.up.pt/up/pt/WEB_BASE.GERA_PAGINA?p_pagina=1001131.

[58] Norm Bridge e Corinne Miller. Orthogonal Defect Classification Using Defect Data to Improve Software Development. 1998.

[59] Kshirasagar Naik e Priyadarshi Tripathy. Basic Concepts And Preliminaries. Em *Software Testing and Quality Assurance, Theory and Practice*, chapter 1.

[60] Software QA Center e Testing Resource. What is API Testing?, 2008. URL: http://sqa.fyicenter.com/FAQ/Testing-Techniques/What_is_API_Testing_.html.

[61] Ana Barbosa, Ana C R Paiva, e José Creissac Campos. Test case generation from mutated task models. *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (EICS'11)*, páginas 175–184, 2011.

[62] Ana C. R. Paiva, João C. P. Faria, Nikolai Tillmann, e Raul A. M. Vidal. A Model-to-implementation Mapping Tool for Automated Model-based GUI Testing. *7th International Conference on Formal Engineering Methods (ICFEM '05)*, 2005.

[63] A. Isabella e Emi Retna. Study Paper on Test Case Generation for GUI Based Testing. *International Journal of Software Engineering and Applications (IJSEA)*, 2012.

[64] Ana C. R. Paiva, João C. P. Faria, Nikolai Tillmann, e Raul A. M. Vidal. Modeling and Testing Hierarchical GUIs. *Abstract State Machines*, 2005.

[65] Duc Hoai Nguyen, Paul Strooper, e Jörn Guy Süß. Automated Functionality Testing Through GUIs. *Proceeding ACSC '10 Proceedings of the Thirty-Third Australasian Conference on Computer Science*, 2010.

[66] Ian Sommerville. Software Documentation. *Software Engineering, vol2: The supporting Processes.*, 2001.

[67] Gurpreet Singh. What is API documentation?, 2012. URL: http://technicalwritingtoolbox.com/2012/03/27/what_is_api_documentation/.

[68] Aline Pires Vieira de Vasconcelos e Cláudia Maria Lima Werner. *Uma Abordagem Para Recuperação De Arquitetura De Software Visando Sua Reutilização Em Domínios Específicos*. Tese de doutoramento, 2004.